Dongsheng Li · Jianxun Lian
Le Zhang · Kan Ren
Tun Lu · Tao Wu
Xing Xie

Recommender Systems

Frontiers and Practices

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn
Springer

Dongsheng Li, Jianxun Lian, Le Zhang, Kan Ren, Tun Lu, Tao Wu and Xing Xie

Recommender Systems
Frontiers and Practices

Dongsheng Li
Microsoft Research Asia, Shanghai, China

Jianxun Lian
Microsoft Research Asia, Beijing, China

Le Zhang
Standard Chartered (Singapore), Singapore, Singapore

Kan Ren
ShanghaiTech University, Shanghai, China

Tun Lu
School of Computer Science, Fudan University, Shanghai, China

Tao Wu
Microsoft, Cambridge, MA, USA

Xing Xie
Microsoft Research Asia, Beijing, China

ISBN 978-981-99-8963-8 e-ISBN 978-981-99-8964-5
https://doi.org/10.1007/978-981-99-8964-5

Jointly published with Publishing House of Electronics Industry, Beijing, China.
The print edition is not for sale in the mainland of China. Customers from the mainland of China please order the print book from: Publishing House of Electronics Industry, Beijing, China.
ISBN of the Co-Publisher's edition: 978-7-121-43508-9

Jointly published with Publishing House of Electronic Industry, Beijing, China

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Foreword

Recommender Systems: Navigators in the Ocean of Information

In 2020, the total global data storage exceeded 40ZB. It is expected that this number will reach 200ZB between 2025 and 2026. Faced with such a huge amount of data, the first challenge of the big data era is how to solve the problem of information overload, that is, how to help users find the content they need or like in the ocean of information. We have seen different types of "information intermediaries," like navigation websites (such as hao123), portal websites (such as Sohu News), search engines (such as Baidu), and recommender systems that this book will introduce. Readers may feel that search engines play the most important role in information acquisition, but in fact, the vast majority of information we passively obtain or seemingly actively but actually passively obtain comes from recommender systems, and this information accounts for the largest share of our information acquisition from the Internet. For example, although we sometimes browse videos of our interests on TikTok, Kuaishou, and Xiaohongshu, most of the time, when we swipe the screen, the new videos come from the recommender system. We may think that the long videos we watch come from accurate grasp of our interests and active positioning of the content, but in fact, over 2/3 of clicks on Netflix come from recommendations, and over 1/2 of clicks on iQIYI come from recommendations. There are also news recommendations on Toutiao, product recommendations on Taobao… We have been tightly wrapped by recommender systems, but this layer of wrapping is very soft, and we often do not realize it ourselves.

The book in the reader's hands is a comprehensive introduction to recommender systems, from theory and methodology to practice. It is authored by internationally renowned scholars in the field. It must be said that there are already many high-quality surveys and monographs on recommender systems, each with its own characteristics. However, most surveys only focus on a certain type of method (such as Adomavicious and Tuzhilin mainly focusing on collaborative filtering,[1] while our survey mainly focuses on the methods of physics[2]) or only delve into a certain problem (such as Herlocker et al.'s survey mainly focusing on how to evaluate a recommender system[3]). Francesco Ricci et al. compiled the book

Introduction to Recommender Systems Handbook,[4] which has a great influence, but this book is actually a compilation of several thematic surveys, and it is not narrated in the same language and symbol system from shallow to deep, so it is only suitable for very professional researchers to read. Liang Xiang's *Practical Recommendation Systems* is an entry-level and practical guide for practitioners,[5] but there is not much theoretical discussion. At the same time, the development speed of recommender systems themselves is very fast. The single algorithm represented by collaborative filtering and matrix factorization has been unable to cope with today's large-scale recommender systems. In fact, in the current mainstream recommender system framework, deep learning and feature engineering have already gained prominence,[6] and the once invincible single algorithm (such as user-based and item-based collaborative filtering) has degenerated into an inconspicuous member of several recall algorithms in the cutting-edge recommender system framework. Therefore, some relatively comprehensive works are far away from the forefront of recommender system technology. Overall, this book is a "just right" recommender system monograph, which takes into account both theoretical and practical aspects, and includes classic algorithms and cutting-edge methods.

I have a deep connection with recommender systems. In 2007, I went to the University of Fribourg in Switzerland to pursue my PhD. The first topic I worked on with Professor Yicheng Zhang was recommender systems, and later recommender system and link prediction became the two main directions supporting my doctoral thesis. After returning to China, the first company I participated in founding[7] initially developed recommender systems for e-commerce websites. My wife also worked in practical recommender system development for a long time and was the main person responsible for building the recommender system for iQIYI. My wife and I have an important shared experience, which is that we both worked on location analysis and recommender systems under the guidance of Professor Xing Xie. I have collaborated with Professor Xing Xie on a total of four papers,[8] including three on how to recommend locations that users may be interested in and one on how to use location information to recommend content.

Despite nearly 30 years of development, the field of recommender systems remains vibrant. This is largely due to the fact that recommender

systems play a significant role in our information acquisition activities, which are an important part of modern life and learning. In addition to the application of deep learning frameworks mentioned earlier, the latest technological developments have presented several new challenges, such as how to better design recommender systems in multimedia environments[9] (closely related to the currently popular multi-modal learning), how to incorporate expert knowledge to build "cognitive" recommender systems,[10] and how to design recommender systems under the premise of privacy protection,[11] among others. During the hype of big data, the Bureau of Cyberspace Security and Emerging Technologies (CSET) of United States released a report suggesting a renewed focus on artificial intelligence applications in small data.[12] Designing recommender systems under the conditions of sparse and insufficient data is also a major challenge. Recently, Professor Xing Xie and his colleagues proposed a possible solution to this problem through their research on knowledge graph-based recommender systems.[13] I have recently focused less on recommender algorithm research and instead turned my attention to ethical issues in recommender systems, such as how to avoid narrow vision and even information cocoons[14] caused by excessive personalization—this is actually a natural extension of my doctoral work[15] on solving the apparent diversity-accuracy dilemma of recommender systems.

It can be said that recommender systems are a vibrant field that seamlessly integrates scientific, technical, and industrial practice. The authors of this book are global research scholars in this field, and most of them are based at Microsoft, making them very sensitive to industry demands! I hope that all readers can gain a lot from this book.

Professor, University of Electronic Science Tao Zhou
and Technology of China

Preface

The recommender system was born in the 1990s with the vigorous development of Internet technology. At the beginning of its appearance, it was generally accepted by academia and industry, achieved wide successes in many areas, such as e-commerce, news platform, multi-media content, daily service, social network, advertisement, and marketing, and has gradually become an indispensable part of the Internet. Today, due to the increasingly prosperous mobile Internet and new media, the recommender system plays an irreplaceable role, continuously reducing the difficulty for Internet users to obtain information, and improving the experience of users interacting with the information systems. During recent years, many successful applications have shown that recommender systems are continuously affecting or even changing the way that humans interact with the information world.

The emergence of deep learning has greatly changed the development of recommendation technology, and it is necessary for researchers and technicians in the field of recommender systems to have a deep understanding of deep learning-based recommendation technology. First, the development of technology is usually like a spiral, and recommendation technology is not exceptional. We can often see the shadows of traditional recommendation technologies behind many new methods and technologies, so that it is very important to connect traditional recommendation technologies with recent deep learning-based recommendation technologies. Therefore, this book spends a lot of space introducing classic recommendation technologies. Secondly, recommendation technology is not limited to Internet applications. There are also a large number of recommendation scenarios in our daily lives. Traditional industries can also use recommender systems to reform their business or management. Therefore, this book focuses on introducing the basic technologies that are not application-specific, so that researchers at different stages and technicians in different industries can all benefit from it. Finally, the recommender system is an application-oriented area. In addition to the learning of methods and principles, it is more important to learn how to design and implement industrial-level recommender systems. Therefore, this book presents to readers how to apply the theory into the practice based on the open source project of Microsoft Recommenders.

To allow readers with different backgrounds and from different industries clearly and completely understand the cause and effect of recommendation technology, this book attempts to view recommender systems from a broader perspective. First, this book starts with classic recommendation algorithms, introduces the basic principles and main concepts of the traditional recommendation algorithms, analyzes their advantages and limitations, and lays the foundation for readers to better understand deep learning-based recommendation technology. Then, this book introduces the basic knowledge of deep learning, focuses on deep learning-based recommendation technology, and analyzes the key problems of recommender systems from both theoretical and practical perspectives, so that readers can gain a deeper understanding of the cutting-edge technologies of recommender systems. Finally, this book introduces the practical experience of recommender systems based on Microsoft Recommenders, an open source project of Microsoft. Based on the source code provided in this book, readers can learn the design principles and practical methods of recommendation algorithms in depth, and can quickly build an accurate and efficient recommender system from scratch based on this book.

This book was written by Dongsheng Li, Jianxun Lian, Le Zhang, Kan Ren, Tun Lu, Tao Wu, and Xing Xie. The work division is as follows:

Dongsheng Li wrote parts of Chapters 1, 2, 3, 5, and 7.

Jianxun Lian wrote parts of Chapters 4 and 6.

Le Zhang wrote the majority of Chapters 6.

Kan Ren wrote parts of Chapters 3 and 4.

Tun Lu, Tao Wu, and Xing Xie coordinated, revised, and reviewed the content of all chapters.

In addition to the authors of this book, several students and partners helped us a lot in writing this book, including Jiafeng Xia, Guangping Zhang, Fangye Wang, Yingxu Wang, Zhengyu Yang, Ziyue Li, and Kerong Wang. We sincerely thank all of them for their tremendous help in the writing of this book.

This book is translated from a Chinese version. Dongsheng Li, Le Zhang, Kan Ren, Tao Wu, and He (Simon) Zhao contributed equally to the translation work. We sincerely thank He (Simon) Zhao for his tremendous contributions in the translation of this book. Scott Graham and Jun Ki Min provided valuable assistance in proofreading and reviewing the English

version of this book, and we express our sincere gratitude for their contributions.

We sincerely thank Mr. Yadong Song and Publishing House of Electronics Industry for devoting their attention to this book and for everything they have done for the publication of this book. We also sincerely thank Springer for everything they have done for the publication of this book.

Due to the limited time, some deficiencies of this book are unavoidable, and we will really appreciate if the readers can let us know any of them.

Dongsheng Li
Jianxun Lian
Le Zhang
Kan Ren
Tun Lu
Tao Wu
Xing Xie
Shanghai, China
Beijing, China
Singapore, Singapore
Shanghai, China
Shanghai, China
Cambridge, MA, USA
Beijing, China

Contents

Footnotes

1 ADOMAVICIUS G, TUZHILIN A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 2005 (17): 734.

2 LÜ Linyuan, MEDO Matúš, YEUNG Chi Ho, et al. Recommender systems. Physics Reports, 2012, 519 (1): 1–50.

3 HERLOCKER J L, KONSTAN JA, TERVEEN LG, et al. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, 2004, 22 (1): 5–53.

4 RICCI F, ROKACH L, SHAPIRA B. Introduction to Recommender Systems Handbook. Boston, MA: Springer, 2010.

5 Liang Xiang. Recommender System Practices. Beijing: Posts & Telecom Press, 2021.

6 Zhe Wang. Deep Learning Recommender Systems. Beijing: Publishing House of Electronics Industry, 2020.

7 Now called Percent Technology Group, then called Percent Technology. Our business starting point can be referenced from a book we wrote earlier—Meng Su, Linsen Bai, Tao Zhou. Personalization: The Future of Business. Beijing: China Machine Press, 2012.

8 In 2015, we wrote a short survey titled “Mining Location-based Social Networks: A Predictive Perspective” for IEEE Data Eng. Bull. That same year, we published a paper on location information titled “Content-aware collaborative filtering for location recommendation based on human mobility data” at the ICDM conference. In 2017, we collaborated on a paper published in Physica A titled “Indigenization of urban mobility”, which analyzed a person’s degree of localization using spatial trajectory data. The degree of localization index can be used for recommending geographical locations of interest. In 2018, we published a paper in IEEE TKDE titled “Scalable Content-Aware Collaborative Filtering for Location Recommendation”, which introduced a cost-effective location recommendation algorithm.

9 DELDJOO Y, SCHEDL M, CREMONESI P, et al. Recommender systems leveraging multimedia content. ACM Computing Surveys, 2020, 53 (5): 1–38.

10 BEHESHTI A, YAKHCHI SMOUSAEIRAD S, et al. Towards cognitive recommender systems. Algorithms, 2020, 13 (8): 176.

11 ANELLI V W, BELLI L, DELDJOO Y, et al. Pursuing Privacy in Recommender Systems: the View of Users and Researchers from Regulations to Applications. 15th ACM Conference on Recommender Systems. New York, NY, USA: ACM Press, 2021, 838–841.

12 CSET. Small Data's Big AI Potential, 2021.

13 GUO Q, ZHUANG F, QIN C, et al. A Survey on Knowledge Graph-Based Recommender Systems. IEEE Transactions on Knowledge and Data Engineering (in press).

14 HOU L, PAN X, LIU K Ch, et al. Information Cocoons in Online Navigation. 2021. arXiv: 2109.06589.

15 ZHOU T, KUSCSIK Z, LIU J G, et al. Solving the apparent diversity-accuracy dilemma of recommender systems. PNAS, 2010, 107(10): 4511.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_1

1. Overview of Recommender Systems

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter first introduces the history of the recommender system and the revolutionary changes in the field of recommender systems. Then, this chapter introduces the basic principles of recommender systems, including introducing the basic assumptions of recommendation algorithms from the perspective of machine learning, introducing how to define the recommendation problem in the form of a machine learning problem, and emphatically introducing the deep learning-based paradigm to solve the recommendation problem—"representation learning + interaction function learning". This chapter also gives an overview of the technical architecture of recommender systems, including the differences between small- and medium-scale recommender systems and large-scale recommender systems. Finally, this chapter introduces the main application areas of recommender systems, such as e-commerce, content platforms, etc., and the actual business value brought by recommender systems to these application areas and compares the three main applications in the Internet field—search, advertising, and recommendation, by the differences and connections among them. Starting from industry problems, this chapter summarizes the

differences in the application of recommender systems in different industries and outlines the solutions to different types of problems.

Keywords Overview – History – Revolution – Recommender system paradigm – Application

1.1 History of Recommender Systems

Since the 1980s, with the vigorous development of Internet technologies, many applications using computers to transmit information have emerged in the field of information technology, such as personal websites, chat systems, emails, online forums, etc. These applications brought to users "happiness" and "trouble" at the same time, i.e., the information overload problem [16]. For instance, the total number of websites in the world has exceeded 1.8 billion [22]. For an individual user, if she/he can browse one website per second and browse 24 hours a day, it will take about 57 years to browse all the 1.8 billion websites, which is unacceptable for any user. Therefore, people are in urgent need of a technology that can not only allow users to enjoy the benefits brought by the information era but also effectively avoid the troubles caused by information overload.

In 1987, the researchers at the Massachusetts Institute of Technology and Michigan State University came up with an interesting idea: to design a new type of information sharing system and only distribute relevant information to those who think the information is valuable to them and not to disturb those who think the information is of no value to them [29]. This idea is actually the origin of recommender system. Since then, the research on recommender systems has gradually deepened and brought higher and higher commercial value. For instance, in 2001, Amazon for the first time introduced the recommender system to their e-commerce platform, which brought a substantial increase in sales [28]. In 2006, Netflix held the "Netflix Prize" competition [3], which attracted a large number of researchers to devote themselves to this field, and also promoted the rapid development of important methods such as matrix factorization in the field of recommendation algorithms. In 2007, Turing Award winner Geoffrey Hinton and his collaborators Ruslan Salakhutdinov and Andriy Mnih jointly proposed to solve the recommendation problem using a restricted Boltzmann machine [42], which opened up the research and development of

recommendation algorithms in the era of deep learning. Since then, the research on recommender systems has begun to flourish, and the value of recommender systems has been proven in more and more scenarios.

1.1.1 Content-Based Recommendation Algorithms

In 1990, Jussi Karlgren from Stockholm University proposed the concept of book recommendation [23] in a technical report. Taking book recommendation as an example, Karlgren introduced how to calculate the similarity between books that users have read in the past and their unread books and then recommend new books to users based on the similarity. This idea can be considered as a typical content-based recommendation algorithm. Its basic assumption is that the items that users like in the future should be similar to the items they liked in the past, so we can recommend items to target users by looking for items with similar content. A large number of content-based recommendation algorithms that have emerged since then were based on this assumption.

Generally speaking, content-based recommendation algorithms include the following three key steps [37]:

- **First, user profiling.** A user's interest in content can be expressed in various ways, such as the user's explicit rating of the item (1–5 stars), whether the user purchased the item (1 or 0), and the user's comment on the item (text). For the items that the user has expressed interest in, they need to be aggregated to form the user's interest set.
- **Second, user modeling.** With the set of user interests, it is necessary to model user interests in a computable way. In this step, it is first necessary to describe the set of items that the user is interested in. Since there are many types of items, such as movies, books, products, and music, different types of items need to be described in different ways. For instance, a movie can be described by means of film type, director, actor, language, release time, and movie reviews, and music is usually described by means of music type, singer, release time, and audio attributes.
- **Third, content recommendation.** After describing the items in the user's interest set, the association between items can be calculated, such as the similarity between two movies. For the items that are not included in the user's interest set, we can calculate their average similarity with

the items in the user's interest set and then select the items with the largest average similarity and recommend them to the user.

Based on the above formulation, researchers proposed a variety of recommendation methods to solve the problem of information overload in different applications. For instance, in 1997, Balabanović and Shoham from Stanford University proposed the Fab system [1] for web page recommendation. In the same year, Pazzani et al. from the University of California, Irvine proposed the Syskill & Webert system [36] to recommend interesting web pages for users. In 2002, Billsus and Pazzani from the University of California, Irvine proposed the Daily Learner system [6], which can recommend personalized news for users. These are all pioneering research works on content-based recommendation. However, these early studies had major limitations, such as low accuracy and lack of diversity. Therefore, with the development of recommendation technology, these traditional content-based recommendation algorithms have been less adopted. In recent years, with the fast development of deep learning, content-based recommendation algorithms have also shown new vitality. In many recommendation scenarios that rely heavily on content, such as news recommendation, a large number of recommendation algorithms proposed to leverage neural network-based representation learning technique in content-based recommendation [52]. More discussions on this will be introduced in the subsequent chapters.

1.1.2 Collaborative Filtering-Based Recommendation Algorithms

After content-based recommendation algorithms were proposed, a lot of researchers found that these algorithms had limitations. For instance, the content and quality of two movies with the same genre or even directed by the same director may vary greatly, and users' interests in them may also vary greatly. Recommendations based solely on content will lead to low accuracy. Meanwhile, only recommending items that are with similar content to the ones that users have been interested in the past will also lead to over-concentration of recommended content, leading to low diversity and serendipity. To solve these problems, in 1992, Goldberg et al. [14] from Xerox innovatively proposed the idea of collaborative filtering, that is, a user may have similar interests with some other users (also called "neighbors"), so she/he is likely to like the items that these neighbors are

interested in. Collaborative filtering can be considered as one of the most important concepts in the field of recommendation algorithms, and it has been affecting the research and application of recommendation algorithms since its appearance.

Key Steps in Collaborative Filtering Algorithms

Generally speaking, a collaborative filtering-based recommendation algorithm includes the following three key steps [40].

First, collect users' interest in items, mainly including explicit ratings and implicit feedback. Explicit ratings refer to users' explicit feedback on items, such as movie ratings from 1 to 5 stars, likes and dislikes, etc. Implicit feedback generally only includes the user's positive feedback and lacks negative feedback. For instance, the user's purchase record on an e-commerce platform can be regarded as positive feedback, and the reason for not purchasing the product should not simply be attributed to dislike. It is possible that the user likes the item but did not purchase it on the target e-commerce platform.

Second, based on the user's interests in the items, the neighboring users who are most similar to the target user are discovered through predefined functions or models, and then we can describe the relationship between each user and her/his neighbors. For instance, we can use cosine similarity to measure the correlation between two users based on their explicit ratings, and then we can select the users with the highest similarities with the target user as the set of neighbors.

Third, for the target user, we can obtain a "collective evaluation" from neighbors on items that the user has not interacted with and then recommend items with high predicted ratings to the user. The key here is how to aggregate neighbors' evaluations. Commonly used methods include weighted average and model-based methods, which will be explained in more detail in the subsequent chapters of this book.

Classification of Collaborative Filtering Algorithms

Collaborative filtering is an idea to understand and solve recommendation problems. There are various kinds of recommendation algorithms based on the idea of collaborative filtering, and the successful methods mainly include the following three categories:

(1) **Nearest neighbor-based approaches [**20, 28, 40**]**. For instance, the user-based collaborative filtering method [20] first finds the nearest neighbors for each target user and then estimates the target user's interest in the target item based on the weighted average of the neighbors' ratings on the item. Similarly, the item-based collaborative filtering method [28, 44] finds the nearest neighbors for each item and then makes a weighted average of the similarity between the target item and the items that the user has interacted with in the past to estimate the interest of the target user.

(2) **Matrix factorization approaches [**5, 24, 41, 43**]**. This type of methods was proposed to solve the sparsity problem of user–item interaction data. The rating vectors of users and items are originally very sparse, so the intersection of rating vectors of different users is very small. However, if the rating vectors of users and items are reduced to a low-dimensional space, the problem of data sparsity will be alleviated, because each dimension on the low-dimensional space does not represent an item but represents a class of items. After dimension reduction, a simple dot product operation between the user vector and the item vector can be used to estimate how much the user likes the item. This type of method achieved very good performance [24] in the "Netflix Prize" competition [3] held by Netflix in 2006 and is also widely adopted by today's mainstream recommender systems.

(3) **Deep learning-based recommendation approaches [**42**]**. Although many deep learning-based methods use a more novel way to model user interests, they are essentially based on the interests of similar users to predict the interests of target users. Especially, many deep learning-based methods can be considered as the neural network versions of the traditional collaborative filtering algorithms. For instance, the NeuMF method [19] replaces the dot product-based score function in the classic matrix factorization algorithm with the multi-layer perception.

1.1.3 Deep Learning-Based Recommendation Algorithms

The earliest deep learning-based recommendation algorithm can be traced back to 2007. The famous Turing Award winner Geoffrey Hinton, together

with his collaborators Ruslan Salakhutdinov and Andriy Mnih, proposed a collaborative filtering method using restricted Boltzmann machines. In the following few years, since deep learning has not been generally recognized by the academic community, deep learning-based recommendation algorithms were not very popular then. It was not until 2012 that Geoffrey Hinton and his students Alex Krizhevsky and Ilya Sutskever proposed to use GPUs to train deep neural network models, which greatly improved the accuracy of image classification tasks on ImageNet, and deep learning began to be recognized by the academic community. Since then, the research related to deep learning has exploded, and deep learning-based recommendation algorithms have also begun to flourish. Essentially, deep learning-based recommendation algorithms do not differ from the scope of the previous two kinds of methods. The current deep learning-based recommendation algorithms can also be divided into content-based recommendation and collaborative filtering. Of course, some methods are a combination of the two. However, compared with traditional recommendation algorithms, deep learning has indeed brought a revolutionary breakthrough.

First, deep learning-based recommendation algorithms can model many types of data that traditional algorithms cannot model, aiming to improve the performance of recommendation algorithms by introducing more information. For instance, by introducing the recurrent neural network into the recommendation tasks [51], the sequential information in the user or item interaction sequences can be better modeled, e.g., items with sequential dependencies generally have a sequential purchase order. For text or image data, many research works try to introduce a pre-trained natural language model [52] (such as BERT [11]) or an image feature extraction network [25] (such as ResNet [17]), to leverage rich text or image information to improve the recommendation performance. Rianne van den Berg, from the University of Amsterdam, Netherlands, proposed to use a graph neural network to model the structural information on the bipartite graph of users and items, such as the user’s neighbors, neighbors’ neighbors, etc. The additional information on these graphs helps modeling the user’s interest more accurately, which then improves the accuracy of the recommendation algorithms.

Second, deep learning-based recommendation algorithms try to improve the modeling of the user–item matching function in the traditional method,

aiming to improve the performance of recommendation algorithms by modeling a more complex user–item relationship. For instance, the Wide & Deep method [7] proposed by Heng-Tze Cheng et al. from Google can not only use the Wide model to model the simple linear relationship between users and items but also use the Deep model to model the complex non-linear relationship between users and items, and the combination of the two can more accurately model the relationship between users and items and improve the accuracy of recommendation. The NeuMF method [19] proposed by Xiangnan He et al. is to replace the dot product score function in the classic matrix factorization algorithm with the multi-layer perceptron as the score function. They hoped to improve the accuracy of the model by a multi-layer perceptron with stronger modeling capabilities. In another work, Xiangnan He et al. proposed the NAIS method [18], which uses the attention mechanism in deep learning to learn the similarity between items, replacing the predefined similarity function (such as cosine similarity, etc.) in the traditional methods, thus improving the accuracy of item-based collaborative filtering algorithms.

Finally, the vigorous development of deep learning technologies has also spawned a large number of new research directions for recommendation tasks and has greatly promoted the development of recommendation algorithms. For instance, many research works try to introduce knowledge graph information into the recommendation tasks [49, 50, 53], which can solve the cold-start problem [50] and can explain the recommendation results [53]. Lixin Zou et al. proposed an interactive recommendation algorithm [55] based on deep reinforcement learning, which can capture the user's interest during the interactions between the recommender system and the user, continuously update the user interests based on new data, and then improve the accuracy of interactive recommendations. Raymond Li et al. proposed a dialogue-based recommender system [27] based on the autoencoder method, which can analyze user interests according to the user feedback during the dialogue and then more accurately recommend movies that users are interested in. Caihua Shan et al. proposed a crowdsourcing task recommendation method [45] using deep reinforcement learning, which can simultaneously optimize the goals of crowdsourcing platform, task publishers, and workers in a dynamic environment. This line of research work is also a hot spot in the current recommendation algorithm research.

1.2 Principles of Recommender Systems

1.2.1 Recommender Systems from the Perspective of Machine Learning

From the perspective of machine learning, recommendation algorithms are mainly concerned with two types of problems. One is the regression problem, which is to predict the user's rating of the item. For instance, on a movie website, the recommender system can predict how a user would rate an item from 1 to 5 stars. This problem was studied in depth in the early recommendation algorithms, thanks to the MovieLens dataset [32] released by the GroupLens Laboratory from the University of Minnesota and the "Netflix Prize" competition dataset [3] released by Netflix. The second is the classification problem, which is to predict whether a user likes an item. For instance, the recommender system can predict whether a user will purchase a certain product on an e-commerce platform. This type of problem received more attention after Amazon began to apply recommender system to its e-commerce platform in 2001. In recent years, the research on regression problems has been relatively matured and new research works are relatively few, which is no longer the most popular research direction in the field of recommendation algorithms. Instead, with the development of deep learning technology, classification problems in recommender systems have attracted more researchers' attention.

Whether it is a regression problem or a classification problem, all key steps in recommender systems can be analyzed and solved from the perspective of machine learning, which also benefits from the advancement of machine learning, especially deep learning, in recent years. As shown in Fig. 1.1, from the perspective of machine learning, a complete recommender system mainly includes five key steps: data collection, data preprocessing, recommendation algorithm selection and model training, recommendation performance evaluation, online deployment, and user feedback. Next, we describe each step in more detail.

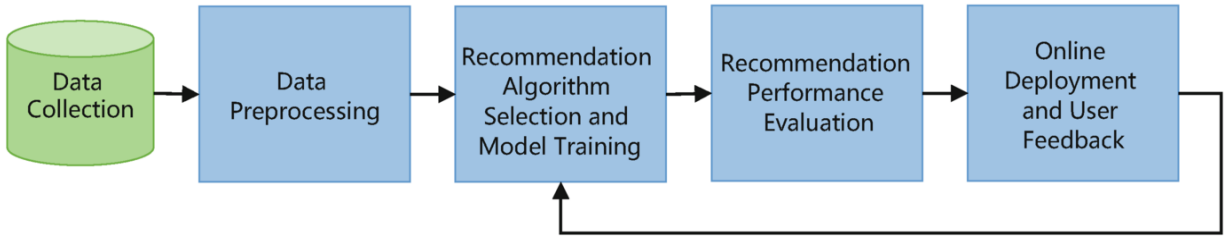


Fig. 1.1 Recommender system from the perspective of machine learning

Data Collection

Data collection is not limited to recommender systems. Any artificial intelligence project needs to collect sufficient data with sufficient quantity and quality to ensure the feasibility of the project. Therefore, data collection is one of the most critical factors that determine the success of a recommender system. In recommender systems, we need to consider the following factors in the process of collecting data.

(1) **The amount of data**. Recommender systems have high flexibility in data requirements. When there is a lot of data, we can directly try complex models, and when there is little data, we can try simple models or design some special interaction methods to acquire data. For instance, in the case of less user interaction data, some content-based recommendation algorithms can be used to make preliminary recommendations. After the users have enough interactions with the system, collaborative filtering methods can be used to make more accurate recommendations. Similarly, a question-and-answer or interactive recommendation algorithm can be used to actively interact with users and continuously collect user information to achieve more accurate recommendations.

(2) **Data quality**. In some recommendation scenarios, such as movie recommendation, the recommender system needs to collect user ratings on items to build user interest sets. The accuracy of user ratings is often affected by many factors, such as score granularity, user memory, even user mood, etc., significantly affecting the accuracy of ratings [9, 26], so it is necessary to consider how to design a more reasonable rating collection mechanism to reduce the noise of rating data. In many recommendation scenarios, such as e-commerce, false purchases or evaluations often occur. These data will seriously

affect the performance of the recommender system, so precautions should also be taken in real recommender systems.

(3) **User privacy**. The security and privacy of user data are increasingly valued by individual users and the society. In September 2021, the "Data Security Law of the People's Republic of China" has come into effect. For recommender systems, in addition to collecting the user's interaction history, it is often necessary to collect the user's personal information to achieve more accurate recommendations. This personal information may include gender, age, occupation, income status, and home address. Once such information is leaked or used illegally, it will bring about serious social problems. Therefore, the recommender system needs to strictly abide by relevant national laws in the process of collecting user data and refer to general user privacy protection experiences, such as the GDPR standard [13] issued by the European Union, to ensure that the user's privacy will not be violated.

Data Preprocessing

After user data are collected, further processing is often required to ensure that the data can be better used to train the model. Common data preprocessing techniques are as follows:

(1) **Outlier processing**. Data collected from the real world often have a lot of noise or missing values, so data noise reduction or missing value imputation may be required. Abnormalities in the range of values generally need to be processed with prior knowledge. For instance, people's age, height, and weight all have common ranges, and values outside the ranges need to be bounded. The abnormality or the absence of some important features will have a greater impact, which can be dealt with in a predictive manner. For instance, we can train a classification model to fill in a user's missing gender information or train a regression model to fill in a user's missing age information.

(2) **Feature engineering**. Before the popularity of deep learning, feature engineering was very important in recommender systems, such as using methods like Singular Value Decomposition (SVD) or Principal Component Analysis (PCA) to reduce the dimensionality [40] of sparse user rating vectors, so as to calculate the relationship between

users or items in low-dimensional space so that similarity is not susceptible to the “curse of dimensionality”. In most deep learning methods, instead of performing feature engineering on the original features, simpler one-hot encoding and other technologies are used to encode category features or user and item IDs, and we can then use one or more layers of neural networks to convert an original feature into a fixed-length vector, which is a commonly used representation learning technique [2] in deep learning. This approach can not only reduce the dimensionality but also embed richer information into the vectorized representations, which is more flexible than SVD or PCA when dealing with various types of data.

(3) **Data analysis**. After the data are obtained, it is usually necessary to conduct some analysis on the data to determine whether there are other problems with the data. For instance, to determine whether the distribution of data is balanced, statistical analysis of data distribution or cluster analysis may be used. Data skewness often exists in recommender systems. For instance, in e-commerce data, the number of items purchased by each user is usually far less than the number of items that she/he has not purchased. In addition, the number of purchases for products often vary a lot, and the number of purchases for popular products may be hundreds of times higher than that of unpopular products in the same category.

Recommendation Algorithm Selection and Model Training

(1) Recommendation Algorithm Selection

The recommendation algorithm is generally considered to be the most important part in the entire recommender system, and the performance of the algorithm often determines the success of a recommender system. We need to pay attention to the following aspects when selecting recommendation algorithms.

a) **Suitability between the algorithm and the data**. Aiming at different types of recommendation problems or data, a more suitable recommendation algorithm can often achieve better results. For instance, for the movie rating prediction tasks, the matrix factorization algorithms have been shown to perform well in the “Netflix Prize”

competition and are therefore the algorithms of choice. However, for news recommendation tasks, content information will play a vital role in the recommendation, so content-based recommendation algorithms are more suitable for tasks such as news recommendation that are highly dependent on content.

b) **The trade-off between efficiency and accuracy**. Generally speaking, complex algorithms can improve the accuracy of recommendations, but the model training efficiency of complex algorithms is usually low. When faced with this kind of problem, it is necessary to weigh and evaluate repeatedly and choose an algorithm with the best accuracy and acceptable computational efficiency.

c) **Ensemble learning**. A single model usually has many limitations in a real recommender system. For instance, the collaborative filtering model is easily affected by cold-start problem. Although content-based recommendation is not affected by cold start, the content analysis is often very difficult. Therefore, in the real system, we can leverage ensemble learning of multiple recommendation algorithms to solve various challenges faced by a single algorithm. Ensemble learning can leverage the complementarity of different recommendation algorithms, theoretically reduce the variance of algorithmic predictions, and improve the generalization ability of the model [54].

(2) Model Training

After selecting the appropriate algorithm, it is necessary to train the recommendation model according to the algorithm and then input the user data into the recommendation model to calculate the recommendation result. The following aspects should be considered here:

a) **Dataset splitting**. Before model training, the data need to be divided into three parts: training set, validation set, and test set. The training set is used to train the model, the validation set is used to evaluate the performance of the trained model, and the test set is used to evaluate the generalization ability of the model on new data which is mainly to prevent overfitting. For the splitting of the three datasets, there is generally no clear standard, and we can choose 80%:10%:10%, 70%:20%:10%, 60%:20%:20%, and other ratios.

b) **Overfitting and underfitting**. Overfitting refers to the case that the model performs very well on the training data but does not perform well on the new data. In contrast, underfitting refers to the case that the model has difficulty in fitting the training data, that is, not performing well enough on the training data. In addition to overfitting caused by improper dataset, overfitting may also result from over-complexity of the model, and underfitting, on the contrary, may result from over-simplification of the model. In the machine learning literature, there are many related techniques that can help to solve these two problems, and this book will not repeat them.

c) **Model update**. The recommender system is an information system that continuously interacts with users, so it will continuously collect new user data. The model can more accurately capture the changes in user interests only when it can continuously use these new data. The model update generally includes offline update and online update. Offline updates are simpler, merging new data with historical data and then retraining the model. Online updates are more complicated and generally require support at the algorithm level. For instance, algorithms such as recurrent neural networks can continuously update the user's interest vector according to changes in input data, and the recommendation results can reflect changes in user interests in real time.

Recommendation Performance Evaluation

(1) Evaluation Metrics

For a recommender system, it is necessary to judge the pros and cons of the system by measuring various metrics. The evaluation of the recommender system generally needs to be considered from two aspects: functional metrics and non-functional metrics.

a) **Functional Metrics**

Accuracy is to measure whether the recommendations made by the recommendation algorithm match the user's interests. Common metrics to measure the accuracy of rating prediction task include mean absolute error (MAE) and root mean square error (RMSE). Common metrics to

measure the accuracy of item ranking tasks include Precision, Recall, F1-score, Normalized Discounted Cumulative Gain (NDCG), etc.

Efficiency is a measure of the computation time and storage space required for the model training and inference process. Efficiency evaluation should be combined with application scenarios. For instance, offline model training is required to be as efficient as possible, but if the update frequency of user data is low, relatively longer offline model training or inference time is also acceptable. For online inference, if the computation delay is high, such as exceeding 100 ms, users may experience significant delays, significantly affecting the user experience.

Diversity and Serendipity. Recommender systems are different from other applications, which need to provide users with personalized information and should always pay attention to the user experience. If the recommended content is too homogeneous, it may cause user boredom. Therefore, recommendations made by recommender systems need to increase diversity and serendipity while ensuring high accuracy. Among them, diversity means that the items in the recommendation list are as dissimilar as possible, and serendipity means that users may not see such items if the recommender system does not recommend to them.

Utility. The proposal of recommender systems is mainly to meet the needs of users for information acquisition and at the same time meet the needs of system designers and other system participants. For instance, from the perspective of an e-commerce platform, the recommender system should promote the interaction between users and the platform, including improving click-through rate, browsing time, quantity of purchased products, sales, etc. Therefore, when evaluating the performance of a recommender system, it is also necessary to consider whether it can meet the requirements of utility.

Interpretability. In many recommendation scenarios, such as healthcare, it is difficult for users to trust the recommender system when they do not understand the reason behind the recommendations. Therefore, it is necessary to provide some explanations to convince users while recommending each item.

b) **Non-functional Metrics**

Security. The recommender system may be attacked by malicious users. For instance, the attacker makes a lot of malicious negative reviews for an item, reducing the possibility of the item being recommended, or making a large number of false positive reviews for an item, increasing the possibility of the item being recommended. To address this kind of problem, it is necessary to design more robust recommendation algorithms to reduce the impact of malicious ratings on the recommendation model. At the same time, it is also necessary to prevent the impact of malicious ratings from the perspective of evaluation mechanism, e.g., increase the cost of false evaluation provided by attackers or detect false evaluation through algorithms.

User privacy. We have already explained the importance of privacy protection in the previous section when discussing user data collection, and we should also consider it in the process of system evaluation, that is, whether the user's privacy may be easily obtained by malicious users or system developers.

Usability. In the process of interacting with users, the recommender system needs to allow users to obtain relevant content conveniently, so the issue of usability needs to be considered in user interactions, for instance, whether the recommended content is displayed in a reasonable position, whether the number of recommended items is too large or too small, and so on. In addition to the above metrics, the recommender system should also pay attention to many system-related metrics, such as scalability, reliability, maintainability, etc. These are similar to the development of other information systems and will not be repeated here.

(2) Evaluation Method

In the recommender system evaluation, in addition to knowing about common evaluation metrics, it is also necessary to know about reasonable evaluation methods. We mainly have the following three evaluation methods for recommender systems: offline evaluation, online evaluation, and user study.

a) **Offline evaluation**. Before the online deployment of recommender system, the collected historical data can be used to evaluate the functional and non-functional metrics of the system, e.g., the accuracy of the recommendations can be evaluated through a separate test set.

However, it may be difficult to measure the system's metrics very accurately from offline evaluation. For instance, when recommending different item lists, the user's response may be different, so in a real scenario, if the recommendation list is changed, the user's decision may not be consistent with the behavior in their historical data.

b) **Online evaluation**. In order to solve the problem of inaccurate offline evaluation, users can be evaluated online. For instance, A/B testing first divides users into two different sets, set A and set B, and uses different algorithms to recommend items to users in the two sets. After a period of time of A/B testing, we can collect the results of online user feedback and compare the pros and cons of the two recommendation algorithms. This evaluation method can solve the aforementioned accuracy problem of offline testing, but it also introduces new problems. Because the online evaluation needs to be carried out in a production environment, if the volume of evaluation is too large or the test is too frequent, the system usability and user experience may be affected. Therefore, the online evaluation needs to be carried out very carefully, and it is generally necessary to do the online evaluation when you have a higher degree of confidence about the new algorithm.

c) **User study**. Both the offline and online evaluations described above can only measure some metrics that are easy to calculate, but to measure many non-functional metrics that are difficult to calculate, a user study is required. There are two ways to conduct the user study: user interviews and questionnaires. The user study can clearly reflect user pain points and even discover key issues that researchers have never considered, which helps researchers find the directions of improvements more quickly.

Online Deployment and User Feedback

After the development of the recommender system, it needs to be deployed online to provide services to users. With the widespread application of the cloud computing technology, the online deployment of recommender systems is increasingly dependent on cloud platforms. The deployment of recommender system based on cloud platform mainly includes two aspects. The first is the online serving of recommendation results. After the

recommendation model is trained, recommendation scores can be obtained for items that the user has not interacted with. These scores need to be stored in a high-performance cloud database, such as Azure Cosmos DB. Using these high-performance cloud databases, online services can efficiently read recommendation results and display them to online users. The second is the online serving of the recommendation model. For the situation that the recommendation scores calculated offline cannot meet the real-time needs of the user, the system needs to calculate the recommendation scores for the user in real time, so the model needs to be deployed online to provide real-time services. In this case, cloud-based machine learning services, such as Azure Kubernetes Service, can be used to deploy the model as an online service to provide real-time recommendation services. The main metric of online services is the response time. Generally speaking, the delay of online services should be less than 100 ms, so that users will not experience obvious delays.

After the online deployment of recommender systems, developers can collect a lot of user feedback from log, which is very important for understanding the system operation status and guiding subsequent algorithm improvement. From the perspective of system operation, these real user feedback can be used to evaluate various functional metrics and some non-functional metrics of the system. For instance, we can analyze the problems in operation and maintenance efficiency, system security, etc., from user logs. Key metrics such as click-through rate and conversion rate of each recommended item can be analyzed from the user's click data, and different types of items can be compared and analyzed. Moreover, we can compare and analyze with historical data to judge whether the operating status of the system has improved. From the perspective of algorithm improvement, developers can analyze user behaviors from logs to find user groups with poor recommendations and then focus on analyzing the reasons for the poor recommendations and finally improve algorithm design to avoid these poor recommendations. It should be noted that in the process of algorithm improvement, there may be a phenomenon of success in one metric but failure in another metric. For instance, some recommendations that were more accurate in the past have become less effective after the algorithm is improved. This is because the optimization goal of the recommendation algorithm is non-convex, so there may be multiple local optimal solutions and the optimal algorithm for some users may not be

optimal for the other users [4]. Therefore, the improvement of the algorithm also needs to be carried out in a personalized manner, that is, it is not necessary to apply the improved algorithm to all users. Users can be grouped, and different recommendation algorithms can be selected for users with different characteristics.

1.2.2 A New Paradigm for Deep Learning-Based Recommender System

The emergence of deep learning has changed many research fields, and the research on recommender systems has also been deeply affected. Compared with traditional machine learning, one of the most important innovations in the field of deep learning is representation learning, which is to represent any kind of information in the form of vectors. For instance, word vector representation (Word2Vec) [31] represents each English word as a vector. In this way, English words that cannot be calculated numerically before can be calculated now. For instance, subtracting the word vector of “man” from the word vector of “king” and adding the word vector of “women”, the vector obtained is very close to the word vector of “queen” [12]. In addition to words, deep learning can represent almost all kinds of information with vectors, such as continuous features, category features, text, image, audio, video, etc., and even contextual information, social network relations, knowledge graphs, etc. can also be represented with vectors. Based on the characteristics of representation learning, the researchers proposed a deep learning-based recommendation algorithm framework, which summarizes a recommendation algorithm into two key steps: representation learning and interaction function learning.

As shown in Fig. 1.2, user representation learning vectorizes user-related data, and item representation learning vectorizes item-related data. After obtaining the vectors of each user and item, the two vectors can be input into the interaction function learning module to calculate the recommendation score and generate a recommendation list based on the scores.

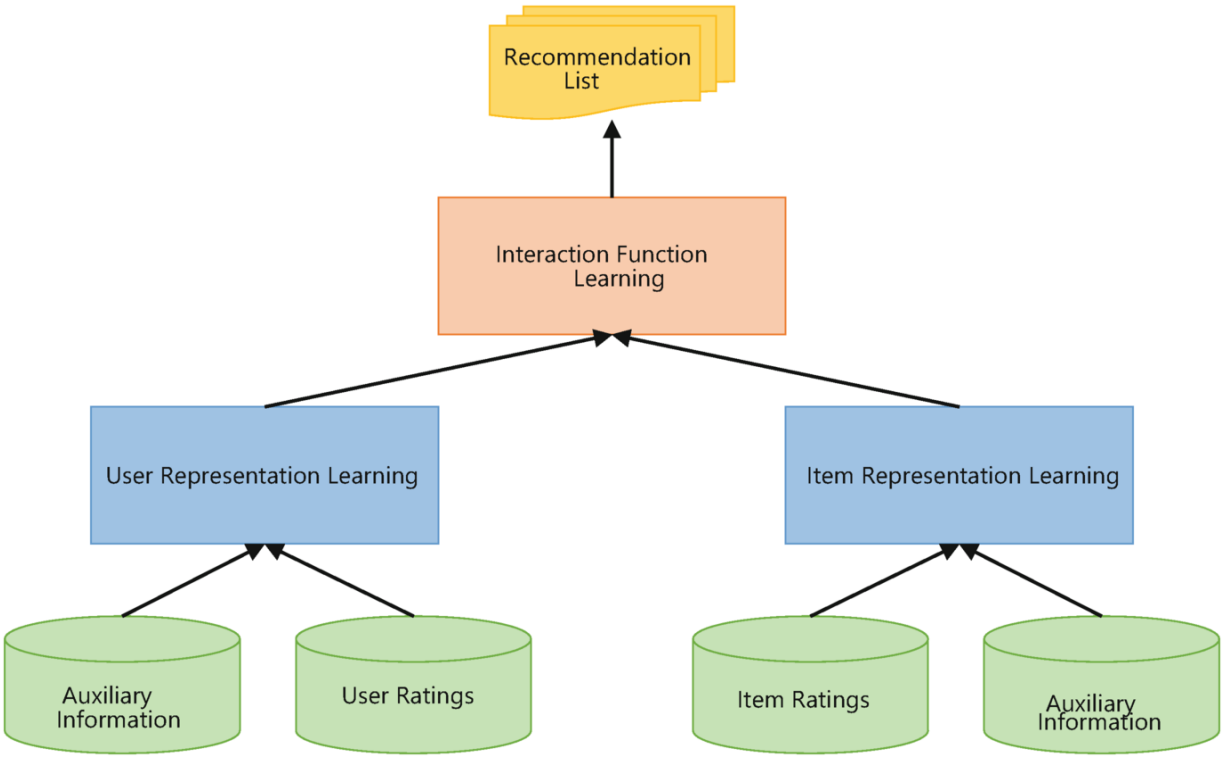


Fig. 1.2 A new paradigm for deep learning-based recommender system: representation learning and interaction function learning

Some of the techniques commonly used in each module are described in detail below.

User Representation Learning

User representation learning can consider all user-related data, such as user ratings, user profiles, user social networks, user reviews, etc. For different types of information, different deep learning techniques need to be used for representation learning. For instance, for structured information such as user ratings and user personal information, multi-layer neural networks or autoencoders can be used to map high-dimensional sparse vectors into low-dimensional dense latent vectors. If it is necessary to model the sequence information clicked by the user, a recurrent neural network can be used to map a sequence to a vector in a low-dimensional hidden space. For the graph data such as the user social network, the graph neural network methods can be used to represent the structural information on the graph as a vector. For text information such as user reviews, pre-trained language models such as Transformer [48] or BERT [11] can be used to represent text as vectors. After learning the representation of different types of information, it is also necessary to fuse the vectors of different information.

The simplest fusion method is to directly concatenate each vector and then perform subsequent processing. A more effective way is through learning, i.e., input different vectors into a new neural network and the output of the neural network can be used as the fused user representation vector.

Item Representation Learning

Item representation learning also needs to consider all item-related data, such as item ratings, item attribute information, item relationship networks, item reviews, etc. For specific representation learning methods, please refer to the previous section (User Representation Learning). Compared with user representation learning, items may contain some unique content information, such as image, audio, and video. For this information, corresponding techniques need to be used for representation learning. For image information, we can use a pre-trained image feature extraction network such as ResNet [17], that is, input an image into ResNet, and then use the features before the prediction layer as the representation of the image. For audio information, audio signal processing methods based on recurrent neural networks such as WaveNet [33] can be used to represent audio data as vectors. The processing of video is relatively difficult. Generally, key frames in the video can be extracted, and then features can be extracted by image processing. Due to the high dimensionality of video features, they are rarely directly extracted and modeled in practical recommender systems.

Interaction Function Learning

After obtaining the vector representations of the user and the item, it is necessary to calculate the possibility of interaction between the user and the item, which needs to be realized through an interaction function. In the classic matrix factorization algorithm, the relationship between the user and the item is modeled by the dot product of the user vector and the item vector. Inspired by it, many deep learning-based recommendation algorithms also use the dot product as the interaction function, mainly because the dot product calculation is very efficient and the recommendation accuracy is relatively high. The dot product is a relatively simple linear multiplication and may not be able to model the complex non-linear relationship between users and items. Therefore, many new studies try to use neural networks with stronger modeling capabilities as the

interaction function, such as the NCF method [19]. Recently, Steffen Rendle et al. [39] compared the two interaction functions of dot product and neural network in more detail. They found that in many scenarios, using the dot product as the interaction function is better than the neural network as the interaction function because the recommendation accuracy will be higher. Of course, since the neural network is a universal approximator that can fit any type of function, the neural network can also approximate the dot product by increasing the depth and the width of the network. However, the amount of calculation and the training cost of the neural network are much greater than the dot product, so it is usually a more reasonable choice to use the dot product as the interaction function in industrial recommender systems [39]. In addition, there are many recommender systems that use the factorization machine [38] as the interaction function. The factorization machine can be regarded as approximating an unknown function in the way of polynomial expansion, so it also has the property of universal approximation. In addition, the complexity of the factorization machine can be manually controlled. For example, only the first-order term and the second-order term are generally used, so its computational complexity will be lower than that of the neural network.

1.2.3 Common Architectures for Recommender Systems

Recommender systems have been applied in many different types of scenarios. Since different application scenarios often face with different problems, small- and medium-scale recommender systems and large-scale recommender systems usually adopt different system designs. In the following, we briefly introduce the common architectures of these two types of recommender systems.

Architecture of Small- and Medium-Scale Recommender Systems

As shown in Fig. 1.3, similar to the common way of using machine learning to solve a practical problem, small- and medium-scale recommender systems mainly include the following five key modules: data processing, recommendation models, model fusion, system evaluation, and online service. Among them, each module needs to be designed according to the business requirements and performance requirements of the system. Due to the small amount of data faced by small- and medium-scale recommender systems, there is no need to consider the challenges of storage and

computation brought by big data, so they can be designed more flexibly. The issues that need to be considered in the design of these models have been introduced before, so we do not repeat them here.

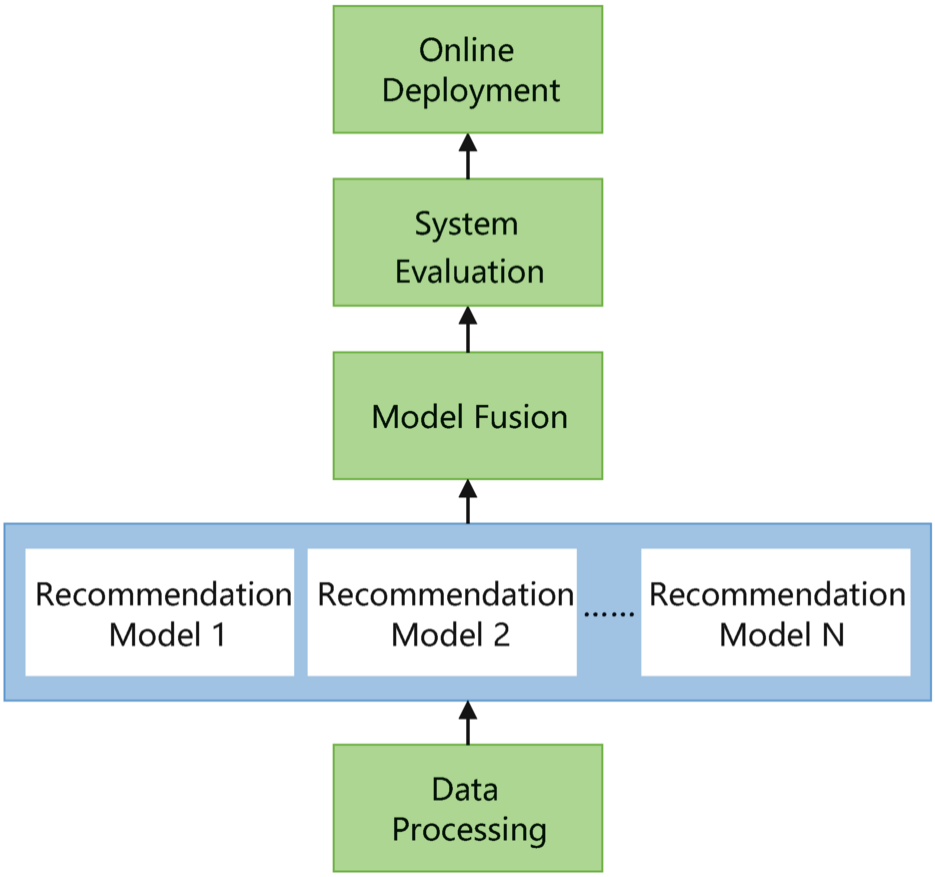


Fig. 1.3 Architecture of small- and medium-scale recommender systems

Architecture of Large-Scale Recommender Systems

Large-scale recommender systems often face with the problem of massive users and massive items, which brings great challenges to the system architecture design and algorithm design. From the perspective of system architecture, mainstream large-scale recommender systems generally adopt a three-layer architecture [47] consisting of offline computing, near-line computing, and online computing, as shown in Fig. 1.4.

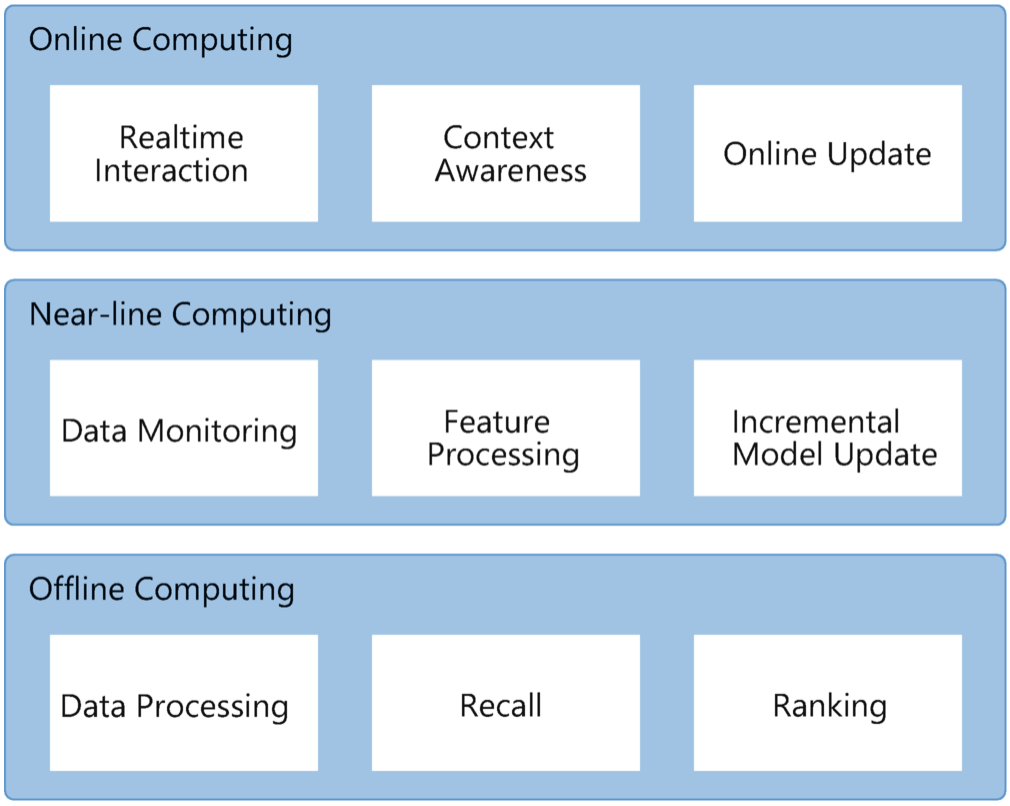


Fig. 1.4 Architecture of large-scale recommender systems

(1) Offline Computing

Offline computing first needs to process all historical data, including data preprocessing, feature engineering, etc. This part is similar to other systems. In addition, due to the huge number of users and items in recommender systems, calculating all the preferences of all users on all items and then performing model fusion will require a very large amount of calculation, which is unacceptable even for offline calculations. Therefore, many systems adopt a two-level structure of “recall + ranking”. In the “recall” module, a small number of items are selected for each user through different recall algorithms (such as popularity, content-based methods, collaborative filtering methods, etc.) and different recall strategies (business rules, business needs, etc.). Then, in the “ranking” module, ranking algorithms such as factorization machines are used to sort these small number of items more accurately. After the ranking module, the list of recommended items for each user can be obtained, and then the offline recommendation list can be stored in the database to provide services for other modules.

(2) Near-Line Computing

Near-line computing can be considered as real-time processing of some offline tasks. However, due to the difficulty of real-time processing, an approximate real-time processing is performed. Near-line computing needs to use real-time online data, but it cannot be guaranteed to be available in real time. This is the most essential difference between near-line computing and online computing. First of all, near-line computing requires real-time feature monitoring of user behavior and online data to obtain some key data currently requested by users. Then, through the feature processing module, the user's real-time data are characterized, and based on the new features, it is decided whether and how to update the user recommendation list. In addition, if the obtained user data need to trigger model updates, such as obtaining new interests of users or new ratings of items, incremental model updates will be performed. When the model is updated, recommendations from the user's current session may reflect that. However, due to the uncertainty of user behavior, the user may leave the current session before the model update is completed, so the model update of near-line computing may not always be obtained by the user.

(3) Online Computing

First of all, the most important function of online computing is to provide users with real-time recommendation services, so the corresponding real-time performance is very important. Generally speaking, if the time delay of an activity exceeds 100 ms [35], it will be clearly perceived by the user, so the service delay provided by the online computing module generally does not exceed 100 ms. Under this limitation, the time complexity, data volume, and the network delay of online computing module cannot be too large. Therefore, general online services use pre-calculated recommendation results, only read on-demand during the online phase, for instance, read the corresponding recommendation list according to the current user request type or a certain business rule, and then return it to the user. Second, the online computing module needs to perceive the user's contextual information, such as the drift of user interests, and then adjust the recommendation list according to the real-time feedback. At the same time, these real-time data also need to be fed back to the near-line computing module and the offline computing module to provide support for subsequent model updates of them. Finally, the online computing module may also

need to perform some online updates to deal with some special scenarios. For instance, after a new user enters the recommender system, the online computing module will collect the user ratings on items and then need to train the model in real time based on the newly obtained ratings. In addition, another important function of online computing is to process related business processes, such as assigning different recommendation results to different users according to the functional logic of the predefined A/B test.

1.3 Values of Recommender Systems

At present, recommender systems have become an indispensable functional component in mainstream e-commerce platforms such as Amazon, Taobao, and JD.com, bringing huge commercial value to these platforms.

1.3.1 Business Values of Recommender Systems

Amazon is one of the earliest companies to commercialize the recommender system, and it is also one of the most successful companies applying the recommender system. 2017 is the 20th anniversary of *IEEE Internet Computing* magazine. The editorial board of the journal decided to select a paper that can stand the test of time from the papers published in the journal. Finally, the editorial board of the journal chose Amazon's recommender system paper—*Amazon.com recommendations: item-to-item collaborative filtering* [28], which is mainly due to the great success of the recommender system in the commercial field and the great influence of this paper to other commercial recommender systems. In 2012, Amazon's total sales increased by about 29% year on year, most of which came from the recommender system [30]. According to the estimation of Microsoft Research, about 30% of the page visits of Amazon website come from its recommender system [46].

The massive success of Amazon's recommender system caught the attention of other companies. Since then, recommender system has gradually become an essential component of e-commerce websites and content platforms. In 2010, a research team from YouTube published a paper introducing their video recommender system [10]. According to the paper, about 60% of the video clicks on the homepage of the YouTube website come from its recommender system. At the same time, the paper

also compares the difference between personalized recommendation and other rule-based recommendations. For instance, the click-through rate of personalized recommendation is 207% higher than that of popularity-based recommendation. Another company that has relied on recommender systems to achieve great success is Netflix. In 2015, Netflix engineers Carlos A. Gomez-Uribe and Neil Hunt published a paper introducing Netflix's recommender system [15]. According to the data given in the paper, about 80% of Netflix's video browsing comes from its recommender system, and the remaining about 20% comes from search engine. The paper also evaluates the commercial value of the recommender system. The recommender system can significantly reduce the user unsubscribe rate. These reduced unsubscribe users generate about 1 billion US$ in revenue for Netflix every year. In addition to Internet companies, the traditional business giant IBM has also successfully applied recommender system to the company's sales business. They use the Cognitive Recommendation Engine [8] developed by researchers from IBM Research to recommend products for customers and help the sales department to improve business. According to their statistics, in 2017, after IBM's sales business adopted recommender system, the sales opportunities for new customers increased by 80%, the winning rate of new customer sales increased by 6%, and the sales opportunities for existing customers increased by more than 2 billion US$ every year.

Chinese Internet companies such as Alibaba, JD.com, and Douban also realized the importance of recommender systems very early on. During Taobao's "Double 11 Shopping Festival" in 2016, Alibaba created approximately 6.7 billion personalized pages for 230,000 merchants on Taobao and Tmall, and the conversion rate of these personalized pages was higher than that of non-personalized pages by about 20% [21]. After Taobao's "Double 11 Shopping Festival" in 2018, Fan Jiang, vice president of Alibaba, made a report at the 2018 Double 11 "Looking at China" forum, in which he mentioned "In this year's Double 11, we can also see that the traffic based on personalized recommendation has exceeded the traffic brought by search engine and other methods, which is a very big change and was completely unimaginable in the past" [56]. In addition, the success of Toutiao, a well-known domestic news platform, is also mainly due to their recommender system. In 2012, Toutiao launched a platform to provide users with news information based on recommendations, gradually

becoming the most popular news portal in China. In addition to applying recommendations to news, ByteDance, the parent company of Toutiao, has also applied the idea of recommendation to many fields such as social media (TikTok), short videos, music, advertising, customer relationship management, and office work. Their recommender system has become ByteDance's core competency.

1.3.2 Recommendation, Search, and Advertising

Recommendation, search, and advertising are called the troika of Internet technology by many people. They are the three most valued technologies in the Internet platform, and they are also the key to the profitability of the Internet platform. From the perspective of the application itself, there are big differences among the three applications, but the three applications have a lot in common in terms of technology, as shown in Table 1.1.

Table 1.1 Comparison of recommendation, search, and advertising

Compared items	Recommendation	Search	Advertising
User interaction	Active & Passive	Active	Passive
Personalization	Strong	Weak	Medium
User acceptance	Strong	Strong	Weak

Application Differences of the Three

From the perspective of application, we can compare the differences between recommendation, search, and advertising through the following three dimensions:

(1) User Interaction

Most recommender systems interact with users in a way that users passively receive information, and a few recommender systems (such as question–answer-based recommender systems) also require users to actively provide some information for the recommender system. Generally speaking, the content provided by the search engine to the user must be consistent with the user's query keywords, so the application must be triggered by the user's active request. Advertisements may cause inconvenience to users, so they often have less interaction with users and only need to be displayed in

a personalized manner based on the user's current context information (such as geographic location, query keywords, etc.).

(2) Personalization
Among the three, recommender systems require the highest level of personalization. Many applications have proved that users usually have unique interests, and recommender systems will fail if they cannot accurately capture users' unique interests. Search engines have weak requirements for personalization, because for users, each query has a clear purpose. If the accuracy of search is reduced in order to introduce personalization, the utility of search engine will be greatly reduced. Advertisements require a high degree of personalization to obtain user clicks, but the advertising platform cannot guarantee to be as personalized as recommender systems due to problems such as real-time response and limited user data, so the level of personalization in online advertisements is between recommendation and search.

(3) User Acceptance
Due to technical limitations, the early recommender systems performed poorly in terms of accuracy and diversity, so user acceptance was not high. At present, with the advancement of technology, the recommender system is more and more popular with users, so its acceptance is close to that of search engines. Due to the limited content of online advertisement, its user acceptance is low, and personalization is the key to solve the low acceptance issue of users in online advertisement.

Technical Similarities Among the Three

Although there are many differences in the application of recommendation, search, and advertising, the three have a lot of similarities in technology, and many related technologies are widely used in the three types of applications. The following is an analysis of the technical similarities among recommendation, search, and advertising from three aspects: data processing, algorithm, and system architecture.

(1) Data Processing
There are two key concepts of users and items in these three types of applications. The documents to be retrieved in the search engine can be

considered as items, and the advertisement itself is a kind of item. Therefore, all three types of applications need to collect user data, item data, and user–item interaction data. For these data, the methods of feature engineering and representation learning are basically similar, so we will not go into detail here. Generally speaking, recommendation, search, and advertisement can use the same data storage method, feature processing process, and representation learning results.

(2) Algorithm

The essential goal of these three types of applications is to find the best matching items for users. In the early days, the algorithms of the three were quite different. For instance, the recommender system used the collaborative filtering algorithm, while the search engine used the PageRank [34] algorithm. However, with the abundance of data and the advancement of technology, the technologies among the three are becoming more and more convergent. At present, mainstream Internet companies have adopted the classic architecture of "recall + ranking" as the algorithmic engine for recommendation, search, and advertising. Through different recall algorithms, a large number of items are filtered to select a small number of candidate items that users may be interested in. Then based on these candidate items, we can use a ranking algorithm to rerank them and finally output the reranked results to the users.

(3) System Architecture

Recommendation, search, and advertising are similar in data and algorithms, which determines their similarity in storage and computing architecture. In addition, all three have high requirements for real-time user interaction, and all need an online module to provide real-time user services. Therefore, by slightly modifying the architecture of the Internet recommender system introduced earlier, it can provide services for a large-scale search engine or online advertising platform.

1.3.3 Industry Applications of Recommender Systems

In addition to e-commerce and the Internet, recommender systems can also be widely used in various industries, such as traditional marketing and sales operations. According to the characteristics of different industries, different considerations need to be made when designing recommender systems. For

instance, for content recommendation, diversity may be an important metric that needs to be considered. In order to improve diversity, the loss of accuracy can often be tolerated; but for medical product recommendation, accuracy is often the most important metric, which cannot be compromised. The following summarizes some unique aspects that need to be considered in designing recommender systems in different industries.

E-Commerce Platforms

The recommender system can be regarded as an essential function of the e-commerce platforms at present, and the related research and technology are relatively mature. First of all, in the e-commerce platforms, the essence of recommender systems is to improve the user experience and enable users to spend less time completing online shopping. Therefore, how to accurately mine the current intentions of users and make targeted recommendations becomes very important. Early e-commerce platforms such as Amazon used item-based collaborative filtering algorithms to find other items that are most similar to the items currently browsing by users and achieved very good results. Second, e-commerce platforms also need to consider the expected benefits of recommended content. Expected revenue needs to be comprehensively measured through multiple aspects such as click-through rate, conversion rate, unit price, profit, etc. Therefore, the optimization goal of the e-commerce platform recommender systems should not be simply the click-through rate. In addition, e-commerce recommendations also need to consider some common sense related to shopping, for example, availability, delivery time, repeated purchases, and whether similar products have been purchased. Failure to take these factors into account may result in recommendations that do not conform to common sense. For example, it is obviously unreasonable to recommend another notebook computer to a user who has just purchased one notebook computer.

Content Platforms

The content platform includes a variety of applications such as video, music, books, news, etc. First of all, for the content recommendation system, how to model the content information into the recommender system is a problem that must be paid attention to. For instance, for news recommendation, the text of the news is crucial to the recommendation, and a good text processing model may significantly improve the quality of the

recommendation. In addition, for different types of content information, different types of modeling methods are required. For instance, a pre-trained language model can be used for text, a time series method can be used for audio, and an image processing technology can be used for images or videos. In addition, the modeling of content needs to pay attention to efficiency; otherwise it may have a huge impact on computational efficiency. Finally, content recommendation also needs to consider the special needs of users for content, such as diversity. For instance, the user's interest in the content is often not single. If the recommended content only covers a small part of the user's interest, that is, lack of diversity, it will cause the user's boredom in the long run.

Daily Services

The daily service platform includes various information related to the basic necessities of life, food, housing, and transportation and focuses on providing users with timely and convenient services, so time and location information often needs to be considered. For instance, when recommending a restaurant, it is necessary to consider whether the restaurant is open at the current time and the distance between the location of the restaurant and the user's current location. When recommending travel routes, it is necessary to consider the traffic sequence between different scenic spots on the route and try to avoid letting users take repeated routes or detours. Therefore, daily service recommendation systems often need to model contextual information. The contextual information mentioned here includes time, geographical location, weather, other surrounding living facilities, and whether there are other users participating together. It is difficult to uniformly describe and model this information. We can refer to general data modeling methods such as one-hot encoding and representation learning and adopt corresponding modeling methods for different types of data.

Social Networks

The social network platform mainly focuses on the social relationship between users and also includes some content information. For instance, on LinkedIn, some users also share some articles written by themselves. In the recommender system of this type of application, it is necessary to consider how to model the information of the user's social network. Generally, a

graph neural network or other recommendation methods based on graph algorithms can be used. For information other than graph features, such as user-created articles, user-uploaded music, interactions between users, etc., this information is modeled in the same way as other types of recommender systems. In addition, social network relationships can be used as auxiliary information to help developers better recommend content. For instance, when recommending videos, the videos watched by the user's friends are more likely to be liked by the user. Therefore, social networks can be used to assist in recommending content and improve the accuracy of recommendation.

Marketing and Sales

Marketing and sales are faced by all commercial companies, and recommender systems can help the marketing and sales personnel of commercial companies find better sales opportunities. For marketing, such as e-mail marketing, it is necessary to maximize the revenue at a given cost. Recommender systems can more accurately predict the interests of users and introduce products that users are more likely to be interested in marketing emails, which can significantly improve the effectiveness of marketing. Similarly, for salespeople, finding sales opportunities can be accomplished with the help of recommender systems. When recommending products to customers, it is necessary to pay attention to the explainability of the recommended products because sales personnel need to communicate with customers, and these explanations for recommended products can often improve the success rate of sales. From a technical point of view, there is no essential difference between the recommender system for marketing or sales personnel and the recommender system for platforms such as e-commerce, so the same algorithm and system architecture can be used.

1.4 Summary

This chapter first introduces the history of the recommender system, including some key events from the introduction of the recommender system concept to the present, such as the emergence of the recommender system concept, content-based recommendation, collaborative filtering, matrix factorization, and deep learning, and the revolutionary changes in

the field of recommender systems. Then, this chapter introduces the basic principles of recommender systems, including introducing the basic assumptions of recommendation algorithms from the perspective of machine learning, introducing how to define the recommendation problem in the form of a machine learning problem, and emphatically introducing the deep learning-based paradigm to solve the recommendation problem—representation learning + interaction function learning. This chapter also gives an overview of the technical architecture of recommender systems, including the differences between small- and medium-scale recommender systems and large-scale recommender systems. Finally, this chapter introduces the main application areas of recommender systems, such as e-commerce, content platforms, etc., and the actual business value brought by recommender systems to these application areas and compares the three main applications in the Internet field—search, advertising, and recommendation, by the differences and connections among them. Starting from industry problems, this chapter summarizes the differences in the application of recommender systems in different industries and outlines the solutions to different types of problems.

References

1. Marko Balabanović and Yoav Shoham. “Fab: Content-Based, Collaborative Recommendation”. In: *Communications of the ACM* 40.3 (Mar. 1997), pp. 66–72. ISSN: 0001-0782. DOI: https://doi.org/10.1145/245108.245124.
2. Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation Learning: A Review and New Perspectives”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828. DOI: https://doi.org/10.1109/TPAMI.2013.50.
3. James Bennett et al. “KDD Cup and Workshop 2007”. In: *SIGKDD Explorations Newsletter* 9.2 (Dec. 2007), pp. 51–52. ISSN: 1931-0145. DOI: https://doi.org/10.1145/1345448.1345459.
4. Alex Beutel et al. “Beyond Globally Optimal: Focused Learning for Improved Recommendations”. In: *Proceedings of the 26th International Conference on World Wide Web*. WWW ’17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 203–212. ISBN: 9781450349130. DOI: https://doi.org/10.1145/3038912.3052713.
5. Daniel Billsus and Michael J. Pazzani. “Learning Collaborative Information Filters”. In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML ’98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 46–54. ISBN: 1558605568.
6. Daniel Billsus and Michael J. Pazzani. “User Modeling for Adaptive News Access”. In: User Modeling and User-Adapted Interaction 10.2–3 (Feb. 2000), pp. 147–180. ISSN: 0924-1868. DOI:

https://doi.org/10.1023/A:1026501525781.

7. Heng-Tze Cheng et al. “Wide & Deep Learning for Recommender Systems”. In: *Proceedings of the 1st workshop on deep learning for recommender systems*. DLRS 2016. Boston, MA, USA: Association for Computing Machinery, 2016, pp. 7–10. ISBN: 9781450347952. DOI: https://doi.org/10.1145/2988450.2988454.

8. *Cognitive Recommendation Engine (CoRE)*. URL: https://cdn2.hubspot.net/hubfs/480025/Project%20--%20Data%20Analytics%20--%20IBM%20--%20Cognitive%20Recommendation%20Engine%20(CoRE)%20--%20Main%20.pdf.

9. Dan Cosley et al. “Is Seeing Believing? How Recommender System Interfaces Affect Users’ Opinions”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’03. Ft. Lauderdale, Florida, USA: Association for Computing Machinery, 2003, pp. 585–592. ISBN: 1581136307. DOI: https://doi.org/10.1145/642611.642713.

10. James Davidson et al. “The YouTube Video Recommendation System”. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys ’10. Barcelona, Spain: Association for Computing Machinery, 2010, pp. 293–296. ISBN: 9781605589060. DOI: https://doi.org/10.1145/1864708.1864770.

11. Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–186. DOI: https://doi.org/10.18653/v1/N19--1423. URL: https://aclanthology.org/N19-1423.

12. Kawin Ethayarajh, David Duvenaud, and Graeme Hirst. “Towards Understanding Linear Word Analogies”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3253–3262. DOI: https://doi.org/10.18653/v1/P19--1315. URL: https://aclanthology.org/P19-1315.

13. *General Data Protection Regulation (GDPR) Compliance Guidelines*. URL: https://gdpr.eu/.

14. David Goldberg et al. “Using Collaborative Filtering to Weave an Information Tapestry”. In: *Communications of the ACM* 35.12 (Dec. 1992), pp. 61–70. ISSN: 0001-0782. DOI: https://doi.org/10.1145/138859.138867.

15. Carlos A. Gomez-Uribe and Neil Hunt. “The Netflix Recommender System: Algorithms, Business Value, and Innovation”. In: *ACM Transactions on Management Information Systems* 6.4 (Dec. 2016). ISSN: 2158-656X. DOI: https://doi.org/10.1145/2843948.

16. B M Gross. *The Managing of Organizations: The Administrative Struggle*. Free Press of Glencoe, 1964.

17. Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: https://doi.org/10.1109/CVPR.2016.90.

18. Xiangnan He et al. “NAIS: Neural Attentive Item Similarity Model for Recommendation”. In: *IEEE Transactions on Knowledge and Data Engineering* 30.12 (2018), pp. 2354–2366. DOI:

https://doi.org/10.1109/TKDE.2018.2831682.

19. Xiangnan He et al. “Neural Collaborative Filtering”. In: *Proceedings of the 26th International Conference on World Wide Web*. WWW ’17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 173–182. ISBN: 9781450349130. DOI: https://doi.org/10.1145/3038912.3052569.

20. Jonathan L. Herlocker et al. “An Algorithmic Framework for Performing Collaborative Filtering”. In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’99. Berkeley, California, USA: Association for Computing Machinery, 1999, pp. 230–237. ISBN: 1581130961. DOI: https://doi.org/10.1145/312624.312682.

21. *How Alibaba Uses Artificial Intelligence to Change the Way We Shop*. Inside Retail. June 7, 2017. URL: https://insideretail.asia/2017/06/07/how-alibaba-uses-artificial-intelligence-tochange-the-way-we-shop/.

22. *Internet Live Stats—Internet Usage & Social Media Statistics*.URL: https://www.internetlivestats.com/.

23. Jussi Karlgren. *An algebra for recommendations: Using reader data as a basis for measuring document proximity*. Tech. rep. 179. Stockholm University, 1990. URL: https://www.diva-portal.org/smash/get/diva2:931533/FULLTEXT01.pdf.

24. Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix Factorization Techniques for Recommender Systems”. In: *Compute*r 42.8 (2009), pp. 30–37. DOI: https://doi.org/10.1109/MC.2009.263.

25. Joonseok Lee et al. “Collaborative Deep Metric Learning for Video Understanding”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 481–490. ISBN: 9781450355520. DOI: https://doi.org/10.1145/3219819.3219856.

26. Dongsheng Li et al. “AdaError: An Adaptive Learning Rate Method for Matrix Approximation-Based Collaborative Filtering”. In: *Proceedings of the 2018 World Wide Web Conference*. WWW ’18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 741–751. ISBN: 9781450356398. DOI: https://doi.org/10.1145/3178876.3186155.

27. Raymond Li et al. “Towards Deep Conversational Recommendations”. In: *Advances in Neural Information Processing Systems 31. 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*. Montréal, Canada: Curran Associates Inc., 2018, pp. 9725–9735.

28. G. Linden, B. Smith, and J. York. “Amazon.com recommendations: item-to-item collaborative filtering”. In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80. DOI: https://doi.org/10.1109/MIC.2003.1167344.

29. Thomas W Malone et al. “Intelligent Information-Sharing Systems”. In: *Commun. ACM* 30.5 (May 1987), pp. 390–402. ISSN: 0001-0782. DOI: https://doi.org/10.1145/22899.22903.

30. JP Mangalindan. *Amazon’s Recommendation Secret*. Fortune. July 30, 2012. URL: https://fortune.com/2012/07/30/amazons-recommendationsecret/.

31. Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: https://doi.org/10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.
32. *MovieLens. GroupLens.*URL: https://grouplens.org/datasets/movielens/.
33. Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. DOI: https://doi.org/10.48550/ARXIV.1609.03499. URL: https://arxiv.org/abs/1609.03499.
34. Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Previous number = SIDLWP- 1999-0120. Stanford InfoLab, Nov. 1999. URL: http://ilpubs.stanford.edu:8090/422/.
35. Lothar Pantel and Lars C. Wolf. "On the Impact of Delay on Real-Time Multiplayer Games". In: *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '02. Miami, Florida, USA: Association for Computing Machinery, 2002, pp. 23–29. ISBN: 1581135122. DOI: https://doi.org/10.1145/507670.507674.
36. Michael Pazzani, Jack Muramatsu, and Daniel Billsus. "Syskill & Webert: Identifying Interesting Web Sites". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence—Volume 1*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 54–61. ISBN: 026251091X.
37. Michael J. Pazzani and Daniel Billsus. "Content-Based Recommendation Systems". In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. ISBN: 978-3-540-72079-9. DOI: https://doi.org/10.1007/978-3-540-72079-9_10.
38. Steffen Rendle. "Factorization Machines". In: *2010 IEEE International Conference on Data Mining*. 2010, pp. 995–1000. DOI: https://doi.org/10.1109/ICDM.2010.127.
39. Steffen Rendle et al. "Neural Collaborative Filtering vs. Matrix Factorization Revisited". In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys '20. Virtual Event, Brazil: Association for Computing Machinery, 2020, pp. 240–248. ISBN: 9781450375832. DOI: https://doi.org/10.1145/3383313.3412488.
40. Francesco Ricci et al. *Recommender Systems Handbook*. Springer, 2011. DOI: https://doi.org/10.1007/978-0-387-85820-3. URL: https://www.springer.com/gp/book/9780387858203.
41. Ruslan Salakhutdinov and Andriy Mnih. "Bayesian Probabilistic Matrix Factorization Using Markov Chain Monte Carlo". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 880–887. ISBN: 9781605582054. DOI: https://doi.org/10.1145/1390156.1390267.
42. Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann Machines for Collaborative Filtering". In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, pp. 791–798. ISBN: 9781595937933. DOI: https://doi.org/10.1145/1273496.1273596.
43. Russ R Salakhutdinov and Andriy Mnih. "Probabilistic Matrix Factorization". In: *Advances in Neural Information Processing Systems 20. 21st Annual Conference on Neural Information Processing Systems 2007*. Ed. by J. Platt et al. Curran Associates, Inc., 2007, pp. 1257–1264.

URL: https://proceedings.neurips.cc/paper/2007/file/d7322ed717dedf1eb4e6e52a37ea7bcd-Paper.pdf.

44. Badrul Sarwar et al. “Item-Based Collaborative Filtering Recommendation Algorithms”. In: *Proceedings of the 10th International Conference on World Wide Web*. WWW ’01. Hong Kong, Hong Kong: Association for Computing Machinery, 2001, pp. 285–295. ISBN: 1581133480. DOI: https://doi.org/10.1145/371920.372071.

45. Caihua Shan et al. “An End-to-End Deep RL Framework for Task Arrangement in Crowdsourcing Platforms”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 49–60. DOI: https://doi.org/10.1109/ICDE48307.2020.00012.

46. Amit Sharma, Jake M. Hofman, and Duncan J. Watts. “Estimating the Causal Impact of Recommendation Systems from Observational Data”. In: *Proceedings of the Sixteenth ACM Conference on Economics and Computation*. EC ’15. Portland, Oregon, USA: Association for Computing Machinery, 2015, pp. 453–470. ISBN: 9781450334105. DOI: https://doi.org/10.1145/2764468.2764488.

47. *System Architectures for Personalization and Recommendation*. Netflix Technology Blog. May 27, 2013. URL: https://netflixtechblog.com/system-architectures-for-personalization-and-recommendatione081aa94b5d8?gi=40ab7aaa227e.

48. Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30. 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, California, USA: Curran Associates, Inc., 2017, pp. 6000–6010. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

49. Hongwei Wang et al. “DKN: Deep Knowledge-Aware Network for News Recommendation”. In: *Proceedings of the 2018 World Wide Web Conference*. WWW ’18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1835–1844. ISBN: 9781450356398. DOI: https://doi.org/10.1145/3178876.3186175.

50. Xiang Wang et al. “KGAT: Knowledge Graph Attention Network for Recommendation”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 950–58. ISBN: 9781450362016. DOI: https://doi.org/10.1145/3292500.3330989.

51. Chao-Yuan Wu et al. “Recurrent Recommender Networks”. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM ’17. Cambridge, United Kingdom: Association for Computing Machinery, 2017, pp. 495–503. ISBN: 9781450346757. DOI: https://doi.org/10.1145/3018661.3018689.

52. Fangzhao Wu et al. “MIND: A Large-scale Dataset for News Recommendation”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 3597–3606. DOI: https://doi.org/10.18653/v1/2020.acl-main.331. URL: https://aclanthology.org/2020.aclmain.331.

53. Yikun Xian et al. “Reinforcement Knowledge Graph Reasoning for Explainable Recommendation”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR’19. Paris, France: Association for Computing Machinery, 2019, pp. 285–294. ISBN: 9781450361729. DOI: https://doi.org/10.1145/3331184.3331203.

54. Zhi-Hua Zhou. “Ensemble Learning”. In: *Encyclopedia of Biometrics*. Ed. by Stan Z. Li and Anil Jain. Boston, MA: Springer US, 2009, pp. 270–273. ISBN: 978-0-387-73003-5. DOI: https://doi.org/10.1007/978-0-387-73003-5_293.

55. Lixin Zou et al. “Neural Interactive Collaborative Filtering”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 749–758. ISBN: 9781450380164. DOI: https://doi.org/10.1145/3397271.3401181.

56. : 11 , Nov. 16, 2018. URL: https://www.sohu.com/a/275896930_114778.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_2

2. Classic Recommendation Algorithms

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter introduces four types of classic recommendation algorithms, including content-based recommendation algorithms, classic collaborative filtering algorithms, matrix factorization methods, and factorization machines. Before the emergence of deep learning, these methods were the most mainstream techniques for recommender systems, widely recognized by both academia and industry. Although after the emergence of deep learning, these technologies are no longer the first choice of the industry, but the basic ideas and practical experience extracted from these technologies still affect the follow-up research. Therefore, in many deep learning-based recommendation algorithms, we can often see the reflections of the above approaches.

Keywords Classic recommendation algorithms – Content-based recommendation – Collaborative filtering – Matrix factorization – Factorization machines

This chapter introduces the recommendation algorithms before the rise of deep learning, including content-based recommendation algorithms and

classic collaborative filtering algorithms. In the content-based recommendation algorithm section, we will focus on how to model both structured and unstructured content. In the classic collaborative filtering algorithm section, three mainstream collaborative filtering methods will be introduced: memory-based methods, matrix factorization methods, and factorization machine methods.

2.1 Content-Based Recommendation Algorithm

The content-based recommendation algorithm [12] is a kind of classic recommendation algorithm, and its concept first appeared in the 1980s. Although it has a long history, it is still widely considered by academia and industry, which is enough to prove its important application value. Different from collaborative filtering algorithms, content-based recommendation algorithms generally only rely on the contents of items and behaviors of users themselves and do not involve the behaviors of other users to make recommendations. Even in the case of cold start (i.e., new users or new items), recommendations can still be made by content-based recommendation algorithms. Therefore, today's commercial recommender systems still use content-based recommendation modules complementary to collaborative filtering recommendations.

Recommender systems infer the user's interests and preferences based on the user's behavior in the system (website, mobile application, etc.) and finally make personalized recommendations for the user. During the entire recommendation process, data may be generated from multiple sources including the user himself, user behavior, candidate item information, and contextual information. In addition to the common numerical data, the types of data used by content-based recommendation algorithms also include text, images, audio, video, etc. As shown in Fig. 2.1, since different data have different formats, the content in the recommender system mainly includes structured data, semi-structured data, and unstructured data.

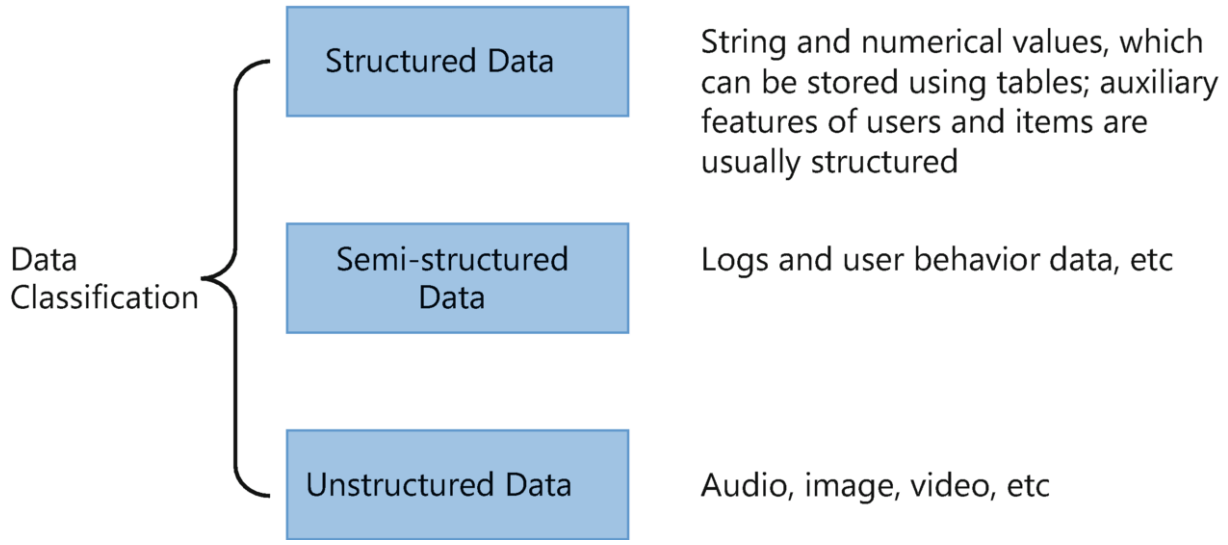


Fig. 2.1 Classification of contents in recommender systems

Structured Data

Structured data can be stored using tables in relational databases. In general, each column in the table represents an attribute or feature, and each row represents a data sample. User attribute data and item attribute data can be stored in different tables, and each attribute of the user and item is represented as a field in the database table, so this type of data is called structured data. Structured data can generally be stored and managed by relational databases such as MySQL and SQL Server and can be queried by a very mature SQL language.

Semi-Structured Data

Semi-structured data does not have the strict structural definition like that in relational databases, but the organization of data is also standardized, such as using predefined tags or rules to separate semantic elements with different meanings in the data or using predefined ways to organize records and fields. This way of defining data structures is also called a self-describing structure. Common data format like XML or JSON belongs to this category. For user behavior in recommender system, the relevant fields are generally recorded in a semi-structured manner, such as using JSON format data to record user online behavior or splitting different fields according to the specified segmentation characters and then splicing them into logs. This type of data is also a kind of semi-structured data. For some

difficult-to-handle semi-structured data, it can also be converted into structured data through preprocessing before processing.

Unstructured Data

The data structure of unstructured data is not clear, even there is no predefined data structure, and it cannot be represented by tables in relational databases nor does it have predefined data specifications like semi-structured data. Common unstructured data include text, images, audio, video, etc. Unstructured data has no fixed data structure, so it is difficult to process it by computer.

This section describes how to design content-based recommendation algorithms for different types of content. Because semi-structured data can usually be transformed into structured or unstructured data, this section will mainly introduce how to design content-based recommendation algorithms for structured and unstructured data.

2.1.1 Recommendations Based on Structured Content

Basic Content-Based Recommendation Algorithms

Basic content-based recommendation algorithms only focus on structured data. In the content-based recommendation algorithm, the most important step is to extract the features of items and users and recommend by calculating the similarity between item feature vectors and user preference vectors. The calculation process of the content-based recommendation algorithm is shown in Fig. 2.2.

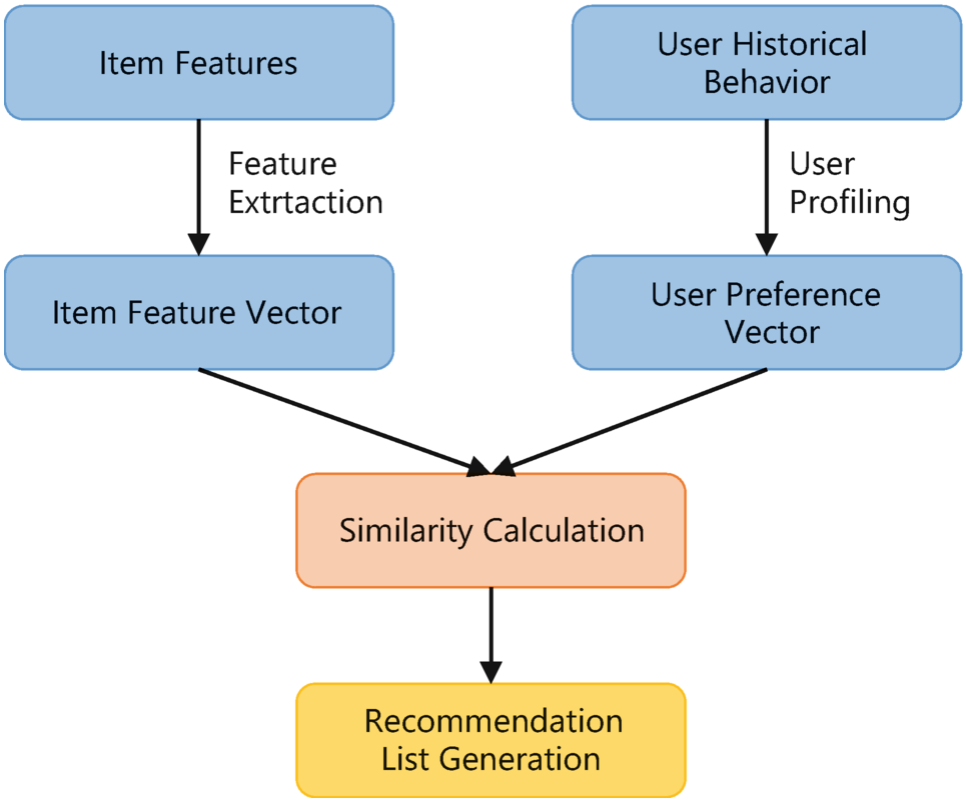


Fig. 2.2 Calculation process of content-based recommendation algorithm

One of the most common similarity calculation method is the cosine similarity, which is defined as follows:

$$\cos(\boldsymbol{F}_u, \boldsymbol{F}_i) = \frac{\boldsymbol{F}_u \cdot \boldsymbol{F}_i}{|\boldsymbol{F}_u|_2 \times |\boldsymbol{F}_i|_2} = \frac{\sum_{k=1}^{K} F_{uk} F_{ik}}{\sqrt{\sum_{k=1}^{K} F_{uk}^2} \sqrt{\sum_{k=1}^{K} F_{ik}^2}}. \tag{2.1}$$

In the above equation, $\boldsymbol{F}_u$ represents the preference feature of a certain user; $\boldsymbol{F}_i$ represents the preference feature of a candidate item; *k* represents the *k*-th feature, and there are K features in the vector. If the value of the cosine similarity is closer to 1, it means that the candidate item is closer to the user's preference. If the value is closer to -1, it means that the candidate item is less suitable for the user.

After calculating the similarity between all candidate items and the user, we can sort the items according to the similarity from high to low and save the Top-*K* candidate items and recommend them to users according to actual recommendation requirements.

Nearest Neighbor Classification Algorithm

K-Nearest Neighbor (KNN) is a very effective and easy-to-master classification algorithm, which is widely used in recommendation algorithms. The main assumption of the algorithm is: in the same feature space, if most of the *K* samples (nearest neighbors) which are most similar to the target sample belong to the same category, the probability that the target sample belongs to this category will also be high.

When classifying, the KNN algorithm only determines the category of the target sample according to the category of the *K* samples closest to the target sample, so the complexity of the algorithm prediction has nothing to do with the total number of training samples, only related to *K*. But the time complexity of the algorithm to find *K* nearest neighbors is related to the total number of samples, for instance, the total computational complexity of obtaining the similarity between any two sample pairs is the proportional to the square of the number of samples. We need to pay attention to three key aspects when using the KNN algorithm: the selection of algorithm hyperparameter *K*, the selection of distance or similarity measurement method, and the rules of classification decision.

Take movie recommendation as an example. When applying the KNN algorithm in the recommender system, we can first find *k* movies rated by the target user that are most similar to the candidate movie. Then we can predict the user's rating on the candidate movie based on the user's rating of the *k* similar movies. Specifically, it mainly includes the following three steps:

(1) **Calculate the similarity**. Calculating similarity is one of the key steps in the KNN algorithm. The similarity or distance commonly used in recommender systems includes: Pearson similarity, cosine similarity, Jaccard similarity, Euclidean distance, etc. Taking the Pearson similarity as an example, its value range is $[-1, 1]$, -1 means that the two users/items are negatively correlated, 0 means that the two are not correlated, and 1 means that the two are positively correlated. We can use $S_{m,n}$ to represent the similarity between item *m* and item *n*, where the similarity calculation process needs to be based on the feature vectors of the two items.

(2) **Select *k* nearest neighbors**. Assuming that the candidate item to be recommended is *m*, we can find *k* items with the highest similarity to

item *m* among all the items rated by user *u* and use $N(u, m)$ to express this collection of *k* items.

(3) **Calculate the prediction score**. After having a set of *k* similar items, the following equation can be used for obtaining the prediction score:

$$\hat{r}_{u,m} = \frac{\sum_{n \in N(u,m)} S_{m,n} \cdot r_{u,n}}{\sum_{n \in N(u,m)} S_{m,n}}. \tag{2.2}$$

Finally, we can sort the candidate items according to the predicted ratings from high and low and recommend the *N* items with the highest predicted ratings to the user.

Relevance Feedback-Based Algorithm

The Rocchio algorithm [8] is a well-known algorithm in the field of information retrieval, which is mainly used to solve the problem of relevance feedback. When using the Rocchio algorithm to construct a user profile vector, it is usually assumed that the correlation between the vector and the features of the items that the user likes is the largest, and the correlation between the vector and the features of the items that the user does not like is the smallest. For instance, if a user gave high scores to the two movies “Your Name” and “Titanic,” then the user’s preference vector can be expressed as {“romance”: 1; “comedy”: 1; “drama”: 0.8}. Afterwards, the user gave a low score to the movie “Good Will Hunting.” At this time, the user preference vector can be updated as {“romance”: 1; “comedy”: 0.5; “drama”: 0.5}.

In content-based recommendation, the Rocchio algorithm can be used to continuously modify the user’s original feature vector to achieve real-time update of the user profile. The feature vector of user *u* is defined as follows:

$$\boldsymbol{w}_u = \frac{1}{|I_r|} \sum_{\boldsymbol{w}_j \in I_r} \boldsymbol{w}_j - \frac{1}{|I_{nr}|} \sum_{\boldsymbol{w}_k \in I_{nr}} \boldsymbol{w}_k. \tag{2.3}$$

In the above equation, I_r and I_{nr} represent the collection of items that the user likes and dislikes, respectively; $\boldsymbol{w}_j$ represents the feature vector of item *j*. The goal of the algorithm is: the new user feature vector is most similar to the feature vectors of the items the user likes and most different from the feature vectors of the items the user dislikes. In practical

applications, the feature vector of the target user may already exist, and we only need to update the user's feature vector as follows:

$$\boldsymbol{w}_u = \alpha \boldsymbol{U}_0 + \beta \frac{1}{|I_r|} \sum_{\boldsymbol{w}_j \in I_r} \boldsymbol{w}_j - \gamma \frac{1}{|I_{nr}|} \sum_{\boldsymbol{w}_k \in I_{nr}} \boldsymbol{w}_k. \tag{2.4}$$

In the above equation, $\boldsymbol{U}_0$ represents the initial feature vector of the user; α, β, γ represent the weight of the initial feature vector, positive feedback, and negative feedback, respectively, which can be set according to experience. For instance, β and γ can be appropriately increased when there are many historical data. In practical applications, we can generally set α to 1, β to 0.8, and γ to 0.2, because the importance of positive feedback is generally greater than that of negative feedback. Figure 2.3 vividly shows the process of user feature vector update.

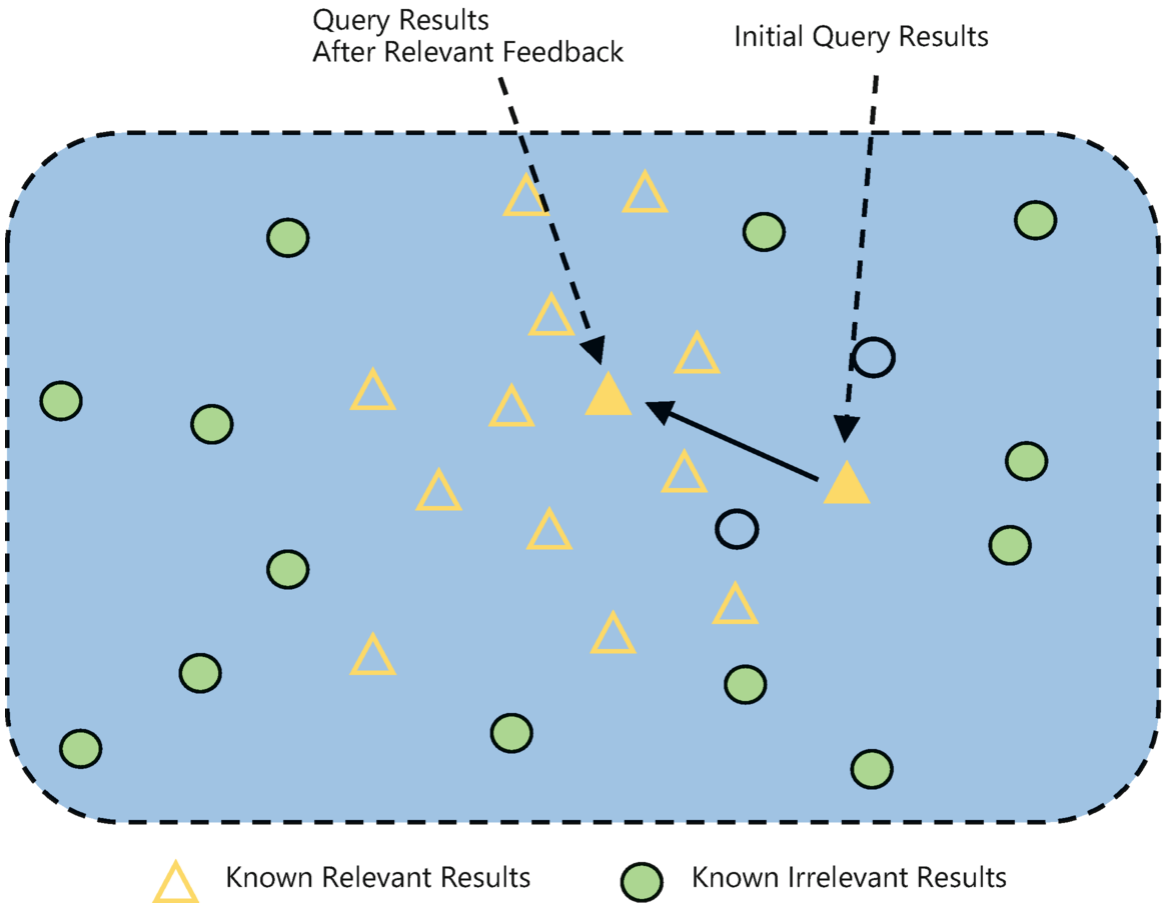


Fig. 2.3 The process of user feature vector update

We can see from the above equation that the Rocchio algorithm is very similar to the average method, except that there is an additional part of

negative feedback, and three weights are set respectively to achieve flexible adjustment. In addition, the Rocchio algorithm also has the advantage that the user feature vector can be updated in real time according to user feedback. Since the update cost is very small, it can be used in real-time recommendation scenarios.

Decision Tree-Based Recommendation

In the content-based recommendation algorithms, another classic algorithm is the decision tree-based algorithm. When the content attributes of the candidate items have good structures, the decision tree usually has an advantage in interpretability than algorithms such as KNN. For instance, a decision tree can show the decision-making process to the user, tell the user the reason why the item is recommended, make it easier for the user to accept the recommendation result, and improve the explainability of the recommendation.

Taking the movie recommendation system as an example, the internal nodes of the decision tree can usually be represented as movie attributes, and these nodes are used to distinguish different types of movies. From the algorithmic perspective, the training of a decision tree is a recursive process, and the conditions for stopping the recursion are: first, the subset of the current node all belongs to the same category, then the splitting ends; second, the attribute values of all samples of the current node have the same value, and the splitting ends; third, the current node has no samples to classify.

The key to decision tree learning is how to choose the optimal partition attribute. Generally speaking, we hope that the branch nodes of the decision tree contain as few classes of samples as possible, that is, the purity of the nodes is higher. Information entropy is one of the most commonly used indicators to measure sample purity. Assuming that the proportion of *k*-th class samples in the current sample set is $p_k(k = 1, 2, \cdots, n)$, then the information entropy of the sample set is as follows:

$$\mathrm{Ent}(D) = -\sum_{k=1}^{n} p_k \log_2 p_k. \tag{2.5}$$

Entropy can measure the uncertainty of variables, and the smaller the value, the more consistent the sample set, that is, the smaller the

uncertainty. When p_k is 0 or 1, there is no uncertainty. Next, we introduce the conditional entropy to describe how to reduce uncertainty by obtaining more information. The conditional entropy defines the mathematical expectation of the information entropy of the conditional probability distribution of *X* on *Y*, as follows:

$$H(Y \mid X) = \sum_{k=1}^{n} p_k H(Y \mid X = x_k). \tag{2.6}$$

Information gain can measure the difference between information entropy and conditional entropy and can be used to select features during the training process of decision tree algorithm. The equation for calculating information gain is as follows:

$$g(D, X) = H(D) - H(D \mid X). \tag{2.7}$$

When selecting features, the feature with the largest information gain is usually selected as the classification feature. Figure 2.4 gives a simple example of a decision tree-based recommendation process. When the system recommends movies for users, it first draws a conclusion based on the user's historical ratings of the movies: when the movie is an action movie, the user is likely to like it; if the movie contains science fiction elements, the user has a high probability to dislike; users may or may not like a movie when it contains romance-related elements.

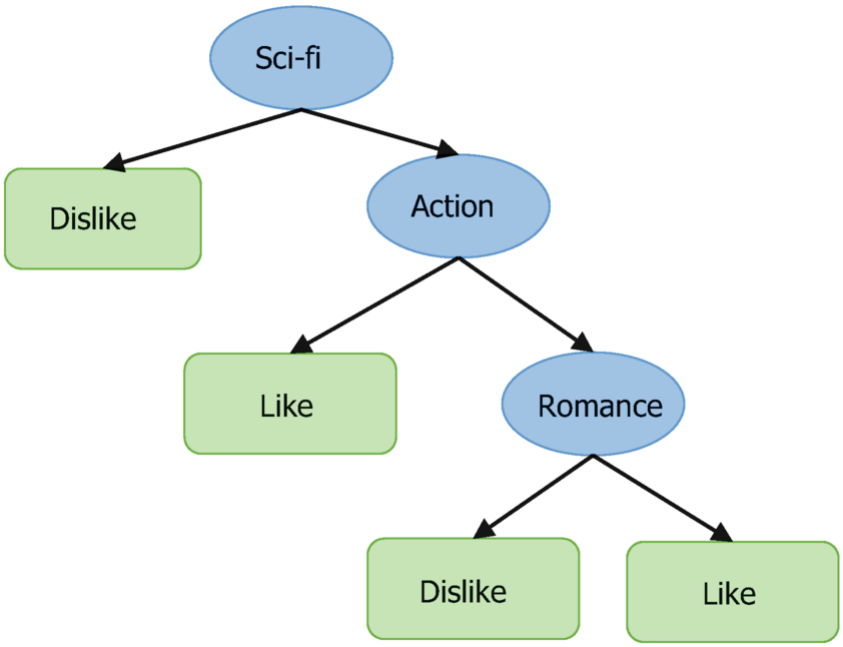


Fig. 2.4 Example of a decision tree-based recommendation process

Naive Bayes Classification

Bayes' theorem is a famous theorem in probability theory, which describes the probability of event *A* occurring under the premise of known condition *B*, generally expressed as $P(A|B)$, *A* and *B* are random events. Bayes' theorem can be expressed by the following equation:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}. \tag{2.8}$$

The classification method based on Bayes' theorem is called Bayesian classification, which is a common classification method, and is also often used in content-based recommendation. The recommendation method based on Bayesian classification can judge whether the user is interested in a candidate item according to the characteristics of the item, such as like or dislike. Suppose the *n*-dimensional vector $F = (f_1, f_2, \cdots, f_n) \subseteq \mathbb{R}^n$ is the feature set of the candidate item, f_i is a feature of the item, and the output space for the recommendation task is $C = \{c_1, c_2, \cdots, c_k\}$. According to Bayes' theorem, it can be expressed that:

$$P(c_k \mid F) = \frac{P(F \mid c_k)P(c_k)}{P(F)}. \tag{2.9}$$

In the above equation, $P(F)$ and $P(c_k)$ represent the prior probability; $P(c_k \mid F)$ represents the posterior probability. Bayesian theorem makes a conditional independence assumption for the conditional probability distribution, that is, different features are independent of each other given a category. Specifically, it can be expressed as:

$$P(F \mid c_k) = P(f_1, f_2, \cdots, f_n \mid c_k) = \prod_{i=1}^{n} P(f_i \mid c_k). \tag{2.10}$$

For a given candidate item, with a feature vector *F*, the posterior probability $P(c_k|F)$ is calculated, and the class with the largest posterior probability is the output. The specific calculation is as follows:

$$y = f(x) = \arg\max_{c_k \in C} \frac{P(c_k) \prod_{i=1}^{n} P(f_i \mid c_k)}{P(F)} \propto \arg\max_{c_k \in C} P(c_k) \prod_{i=1}^{n} P(f_i \mid c_k), \tag{2.11}$$

where the denominator is a constant, which is the same for all categories. Therefore, only the numerator needs to be maximized.

Linear Classification-Based Content Recommendation Algorithm

The content-based recommendation problem can usually be regarded as a classification problem, so various classification methods commonly used in machine learning can be used, such as classical linear classifiers. The goal of a linear classifier is to find a plane in a high-dimensional space to distinguish samples of different classes so that samples of different classes are distributed on different sides of the plane as much as possible. In the recommendation algorithm, this is equivalent to finding a classification boundary to divide items into two categories that users like and dislike. For instance, if a user likes to watch action movies, then the classification boundary is whether the movie belongs to action movies. In practice, the splitting conditions are more complex, usually a combination of multiple features.

As shown in Fig. 2.5, suppose the feature of the input movie is $\boldsymbol{F} = (f_1, f_2, \cdots, f_n)$, where f_i represents the *i*-th feature of the movie, and the output result Y represents whether the user likes to watch the movie. The linear classification model tries to find the plane $Y = \boldsymbol{W} \cdot \boldsymbol{F} + b$ in the feature space $\boldsymbol{F}$, hoping that this plane can separate the movies that the user likes and dislikes.

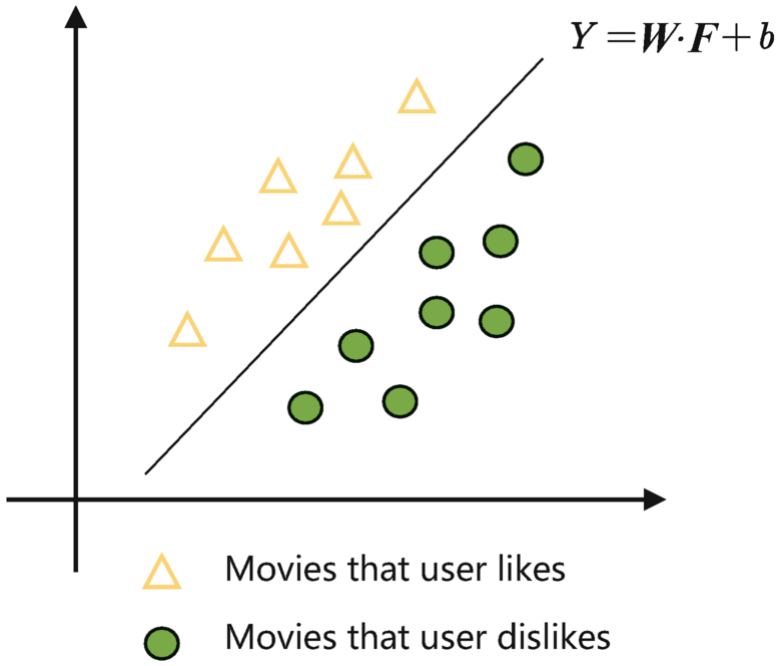


Fig. 2.5 Example of a linear classification model-based recommendation

In $Y = \boldsymbol{W} \cdot \boldsymbol{F} + b$, $\boldsymbol{W}$ represents the weight corresponding to the movie feature, and *b* represents the bias. Both $\boldsymbol{W}$ and *b* are parameters in the model and need to be obtained through learning. A commonly used parameter learning method is the gradient descent method, that is, the parameters are updated in the direction of gradient descent until convergence. The update of each iteration is described as follows:

$$\boldsymbol{W}^{t+1} := \boldsymbol{W}^t - \eta(\boldsymbol{W}^t \cdot \boldsymbol{F} + b^t - Y)\boldsymbol{W}, \tag{2.12}$$

$$b^{t+1} := b^t - \eta(\boldsymbol{W}^t \cdot \boldsymbol{F} + b^t - Y). \tag{2.13}$$

In the above equations, *t* represents the number of iterations; η represents the learning rate, which controls the step size of each iteration of the model update. After continuous iterations to achieve convergence, the corresponding hyperplane is found to classify the movies. For candidate movies, we can judge whether the movie features satisfy the condition $\boldsymbol{W} \cdot \boldsymbol{F} + b > Y$ and then sort and recommend based on the classification results.

2.1.2 Recommendations Based on Unstructured Content

Unstructured data refers to data whose data structure is not clear or not predefined. Common unstructured data include text, image, audio, video,

etc., which are difficult to represent with the table structure in the database. Unstructured information is usually text-heavy but may also contain data such as dates, numbers, and facts as well as multimedia information such as images, audio, and video. This leads to irregularities and ambiguities in unstructured data, which are harder for computer systems to understand than data stored as fields in databases or annotated data (with semantic labels) in files. Although unstructured data has the disadvantages of complex structure, non-standard and high processing overhead, the high data volume and rich information determine that unstructured data is a treasure to be discovered by recommender systems.

Item representation is the basis of recommender systems. Using the item representation vector, the recommender system can easily calculate the item's intra-category similarity and inter-category preference matching, so as to make recommendations. The recommender system's processing of unstructured data also follows this idea. Through representation algorithms, representation learning algorithms, etc., unstructured data is processed into vectors and connected to downstream tasks. All kinds of unstructured data have their own unique representation methods, but the processing ideas are interlinked. This section will focus on the item representation of text data and briefly describe other forms of data processing.

Text Representation

There are two common technical pathways for text representation, one is discrete representation in classical machine learning, and the other is distributed representation in deep learning.

(1) Discrete Representation

a) **One-hot encoding**. One-hot encoding is a binary vector representation of categorical variables, and it is the simplest and most commonly used encoding method when dealing with discrete data. One-hot encoding uses N state registers to encode N states, each register is the state of an encoding, and at any time there is only one state register activated. As shown in Fig. 2.6, the IDs of basketball, football, and rugby are 0, 1, and 2, respectively, corresponding to the 0-th, 1-th, and 2-th bits in the one-hot encoding are 1, that is, the one-hot encodings are [1, 0, 0], [0, 1, 0], and [0, 0, 1], respectively.

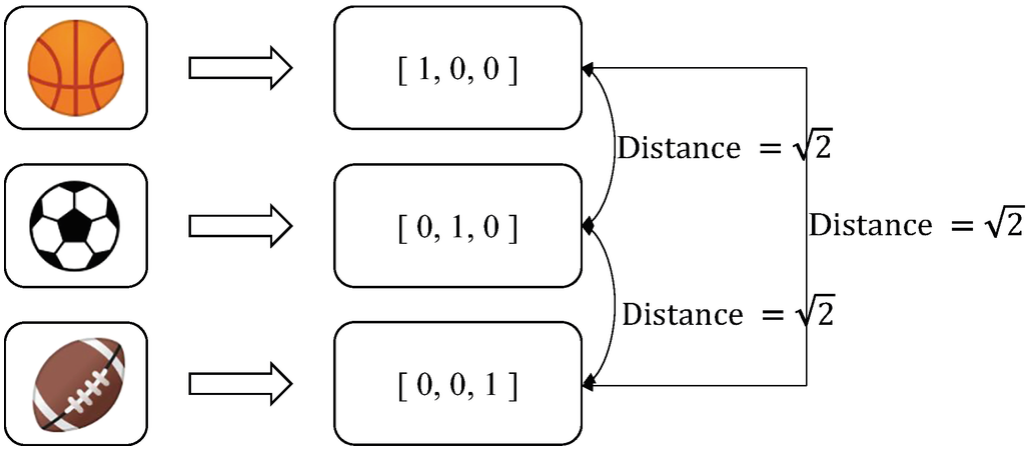


Fig. 2.6 Example of one-hot encoding

In recommender systems, it is very important to measure the distance or similarity between item representations. Commonly used distance or similarity calculations are performed in Euclidean space, and the Euclidean distance between each state representation under one-hot encoding is the same. For mutually independent state categories, this encoding method is more reasonable. However, if the distance can be reasonably calculated by the discrete features themselves, such as numerical features, there is no need for one-hot encoding. In addition, one-hot encoding also requires that each state category is independent of each other. If there is a continuous relationship between states, it is more appropriate to use distributed representation. Finally, the features obtained by one-hot encoding are very sparse, and if the state space is too large, it will bring the curse of dimensionality.

b) **Bag-of-words model**. Bag-of-words (BOW) [8] is a relatively simple language model that converts text into vector representation. As shown in Fig. 2.7, the bag-of-words model regards the text as a collection of all words in the text. It does not consider the order of the words but only considers the number of occurrences of each word in the sentence.

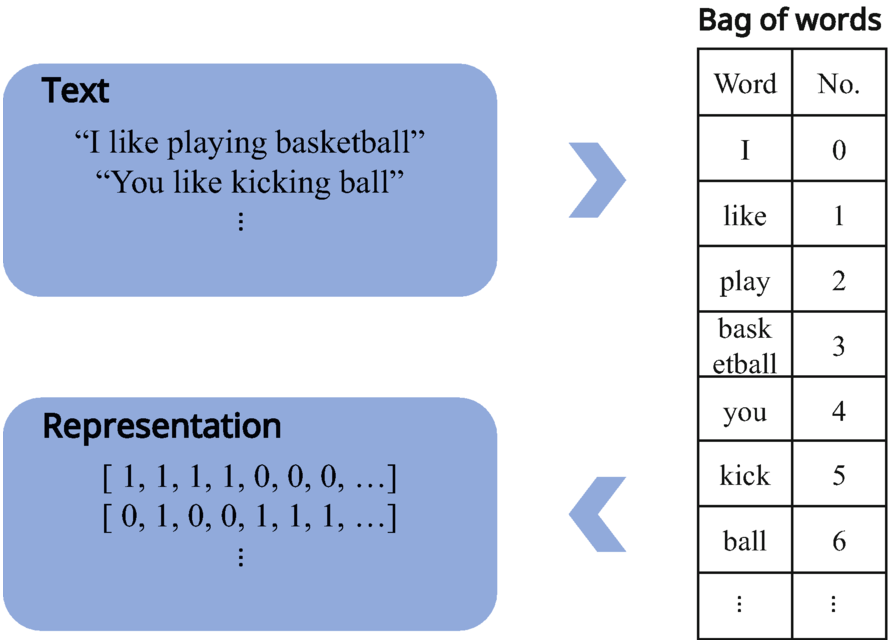


Fig. 2.7 Example of bag of words

The advantages and disadvantages of the bag-of-words model are very obvious. Its advantages are simple and easy to implement. The disadvantages are that it cannot consider the structure and order of the text and its expressive ability is limited.

c) **N-gram model**. N-gram model is an algorithm based on statistical language model, which is an extension of bag-of-words model. This model takes adjacent *N* words as a unit and assumes that the occurrence of the *N*-th word is only related to the previous $N-1$ words (Markov assumption), and not relevant to other earlier words. From the perspective of the N-gram model, the occurrence probability of the entire text is equal to the product of the conditional probabilities of the individual words that make up the text. The conditional probability of N-gram phrases can be approximated by counting the frequency of N-gram phrases in the corpus. Commonly used variants are Bi-gram ($N=2$) and Tri-gram ($N=3$). When $N=1$, the N-gram model degenerates into a bag-of-words model.

As shown in Fig. 2.8, when using the N-gram model for text representation, first, the content in the text is operated by a sliding window with a size of N, after which a sequence of fragments with length *N* is formed, and each fragment is called a gram. Then, we can count the occurrence frequency of all grams and filter according to the predefined threshold to form a key gram list, which is also the feature

vector space of the text, and each gram in the list is one dimension of the feature vector.

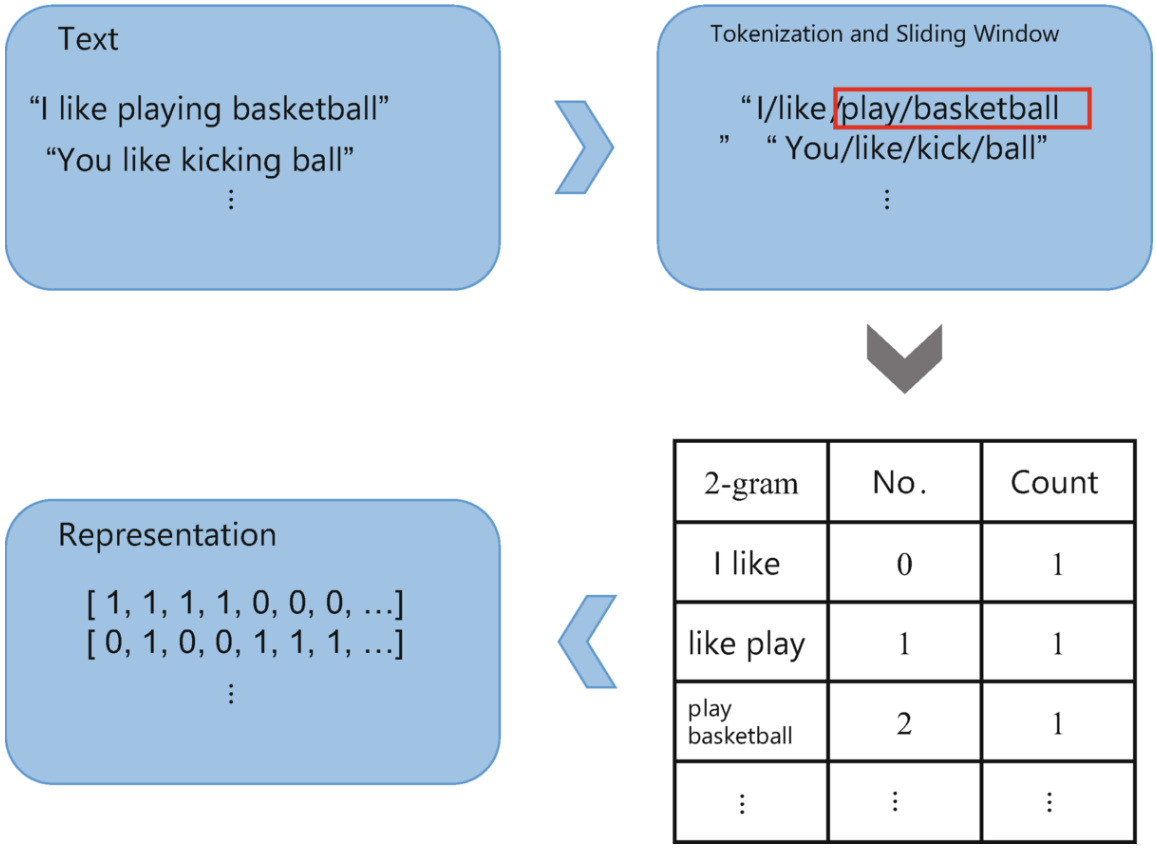


Fig. 2.8 Example of N-gram model

The advantage of the N-gram model is that it models the local sequence information of the text and solves the problem of different semantic meanings caused by different word sequences. For instance, “I like to play basketball” and “Basketball likes to play with me” will produce the same textual representation under the bag-of-words model, while the N-gram model can distinguish them. The disadvantage of the N-gram model is that increasing the length of the phrase will make the total number of N-grams expand exponentially and become more sparse. For instance, for a corpus of 20,000 words, the total number of Bi-grams is 4×10^8, and the total number of Tri-grams reaches 8×10^{12}.

d) **TF-IDF model**. TF-IDF (Term Frequency—Inverse Document Frequency) [8] is an algorithm to evaluate the importance of a word to

a document in the corpus. Its core assumption is that the importance of a word in the document is proportional to the frequency of the word in the text and inversely proportional to the frequency of the word in the entire corpus.

The term frequency (TF) of word *i* to document *j* is defined as follows:

$$\mathrm{TF}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}, \tag{2.14}$$

where $n_{i,j}$ is the number of occurrences of the word *i* in the document *j*, and the denominator is the sum of the occurrence times of all words in the document *j*. Dividing the two can prevent TF from biasing toward long text.

The inverse document frequency (IDF) of a word is defined as follows:

$$\mathrm{IDF}_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}, \tag{2.15}$$

where the numerator is the number of all documents in the corpus, and the denominator is the number of documents containing the word *i* in the corpus.

The TF-IDF of word *i* to document *j* is defined as:

$$\text{TF-IDF}_{i,j} = \mathrm{TF}_{i,j} \times \mathrm{IDF}_i. \tag{2.16}$$

It is not difficult to see from the above equation that words with high importance to a document need to meet the two conditions of high word frequency in the document and low word frequency in other documents in the corpus. The former filters out occasionally used words in the document, and the latter filters common high-frequency words. Finally, it filters out the repeatedly mentioned and topical words in the document.

The advantages of TF-IDF are simple logic, fast calculation and good interpretability. The disadvantage is that the standard for measuring the importance of words is too simple to deal with the inconsistency between word frequency and importance. In addition, TF-IDF ignores the sequential information of words and cannot reflect the relationship and influence between words and context.

(2) Distributed Representation

The idea of distributed representation is to establish a mapping from words to low-dimensional continuous vector space through machine learning, so that semantically similar words are mapped to closer regions in the vector space, while semantically irrelevant words are mapped to farther regions. This property can be used for generalized analysis of words and sentences, and it is a means to achieve the purpose of word semantic speculation and sentence sentiment analysis.

a) **Co-occurrence matrix-based model**. Generally speaking, words with similar semantics often co-occur in the context. This phenomenon provides ideas for modeling the similarity between words. A simple method is to scan all the sentences in the corpus, count the number of times other words appear around each word, construct a word adjacency matrix, and use the value of the corresponding column/row of the word as the vector representation of the word. However, this vector is too large and too sparse, direct use of which will consume a lot of storage and computing resources. Therefore, in practical applications, it is necessary to reduce the dimension of the vector. As shown in Fig. 2.9, the most direct matrix dimensionality reduction method is eigenvalue decomposition or singular value decomposition, which can maximally preserve information in the high-dimensional sparse co-occurrence matrix into the low-dimensional dense embedding matrix by keeping the largest components of the features. After dimensionality reduction, each row or column can still be represented as a vector of the word.

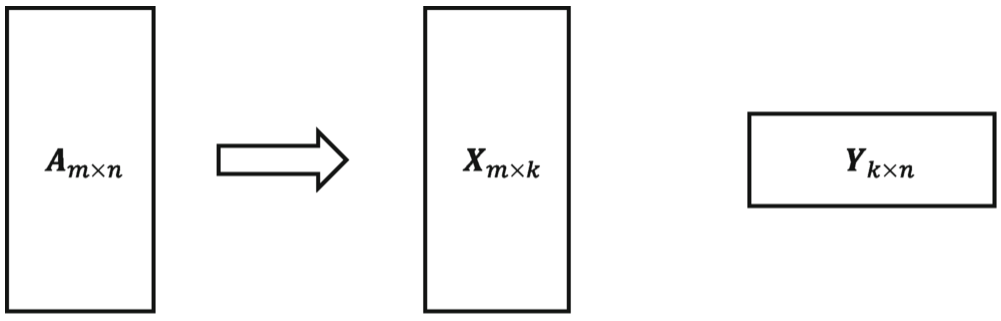


Fig. 2.9 Example of dimensionality reduction for co-occurrence matrix

b) **Neural network-based models**. In distributed representation, deep learning has great advantages over traditional methods and has occupied a dominant position in text representation in recent years. The core idea of deep learning-based text representation methods is to use vectors to represent text, such as the Word2Vec method [9] which uses vectors to represent words and the Paragraph2Vec method [7] which uses vectors to represent paragraphs. After each text is vectorized, their similarity can be efficiently calculated by dot product or cosine similarity, and then recommendations can be performed using content-based recommendation algorithms or item-based collaborative filtering algorithms. Since the Transformer method [17] was proposed, the field of text representation learning has increasingly begun to use pre-training techniques to improve learning capabilities, such as BERT [4], UniLM [5], GPT-3 [2], etc. Through unsupervised or self-supervised training on very large-scale corpora, these pre-trained models can obtain general distributed representation capabilities of natural language. The text representations based on the output of these pre-trained models can also be applied to content-based recommendation algorithms. These technologies will be introduced in detail in subsequent chapters and will not be repeated here.

Representation of Non-text

With the rapid development of technology, the forms of content in the Internet are flourishing, gradually developing from a single text form to a fusion of multimedia multi-modal information, such as images, videos, and audios. The modeling and representation of these multimedia multi-modal information have become the key to improving the performance of recommender systems currently.

(1) Image Representation

Before the rise of deep learning technology, image feature extraction usually relied on manual feature extraction, that is, designing some feature extraction techniques through human experience to extract features of different types of images. The features extracted by these methods can be divided into two categories: one is general features, including pixel-level features (such as the color and position of pixels), local features (summary

of features of some regions on the image), and global features (summary of all features of the image); the other category is domain-related features, which are strongly related to applications, such as face and fingerprint. After the features are extracted, a machine learning model can be trained to obtain the relationship between image features and user preferences and then use these relationships to calculate recommendation scores. For instance, we can consider the image features of the items that the user has interacted with as a representation of the user's interests and then train a classifier to distinguish items that the user likes or dislikes.

Image representation based on deep learning tries to understand the image itself. In the text representation part, methods such as BERT use downstream self-supervised tasks to pre-train the upstream representation model and then migrate the representation model to other downstream tasks such as recommender systems. The image visual representation can also follow this idea. The image is encoded by a task-specific pre-trained representation model, converted into a distributed vector representation, and then recommendations can be performed by a *K*-nearest neighbor-based recommendation algorithm. In terms of pre-training tasks, image representations can be pre-trained using supervised tasks such as image classification or unsupervised tasks such as image generation. Among them, supervised tasks such as image classification have a clear training objective. When the image representation information required for the recommendation task is relatively clear, such as recommendations based on image style preferences, the pre-training of the representation model can be designed and adjusted in a targeted manner, so that it can improve the quality of image representation and recommendation performance. In contrast, if image representation needs to be applied to multiple downstream recommendation tasks, or there is no clear recommendation target, a representation model pre-trained for generative tasks may be more reasonable. In addition, an end-to-end approach can also be used to train image representation and recommendation models at the same time. The advantage of this method is that image representation is directly oriented to downstream recommendation tasks. The disadvantage is that the model is complex and difficult to train.

(2) Video Representation

The representation of video is often performed by characterizing the text associated with the video, e.g., long text such as video title and description and sparse text attributes such as tags. Before the emergence of distributed text representation technology based on deep learning, long text was more used in search engines than recommender systems, and tags were the core of the recommendation tasks at that time. A label is an abstract description of a subject and subjects belonging to the same label share this attribute. The more labels shared by two subjects, the more similar they are. However, video tags are often very sparse, so how to effectively diffuse tags to solve the sparsity issue is the core problem of the recommender system at that time.

An excellent example of solving this problem is the Adsorption algorithm [1] from YouTube. The core of the Adsorption algorithm is the video co-view graph. First, a user--video bipartite graph is constructed, and then edges between videos are generated based on rules such as the number of users who have watched two videos at the same time. Finally, the label adsorption is performed on the generated video relationship graph. In the label adsorption process, each node first calculates its own new label according to the label passed by the neighbor and then propagates the new label back to the neighborhood. During this process, the label gradually diffuses and finally converges, and a stable and smooth distribution is formed on the nodes connected to any of the original nodes.

Based on the Adsorption algorithm, the timeliness of user behavior is further considered, “co-viewed” can be restricted to “co-viewed in one session of the user.” The similarity of two videos is calculated as follows:

$$r(v_i, v_j) = \frac{c_{ij}}{f(v_i, v_j)}, \tag{2.17}$$

where c_{ij} represents the number of times a video is co-viewed in all sessions; $f(v_i, v_j)$ represents a regularization function that tries to punish the popularity of videos, in which a simple solution is to multiply the number of times the two videos have been viewed. So far, we have seen the prototype of collaborative filtering.

Although deep learning technology has made great progress, but due to factors such as computation costs, most current video recommendations in the industry rely on technologies such as long text representations, tags, and social recommendations. In addition, label-based methods are

computationally simple and thus are still widely used in the “recall” part of large-scale recommender systems.

(3) Audio Representation

There are also two ways to represent an audio: by means of associated text and for the audio itself. Taking music representation as an example, music metadata can be divided into three categories: editorial metadata (some labels on the music claimed by the music publisher), cultural metadata (song listening statistics, co-occurrence relationship, etc.), and acoustic metadata (analysis of music audio signals, such as beat, tempo, pitch, instrument, mood, etc.). The first two types of metadata are presented in the form of tags and long text, respectively, and can be calculated and utilized using the tag propagation and text representation methods introduced before. In terms of analyzing music audio signals, Query by Singing and Humming (QBSH) system is an important technology for audio retrieval using audio signals before deep learning. This technology extracts information from audio signals, compares them with those in the database, and then sorts and retrieves based on similarities.

There are three key parts in query by singing and humming: onset detection, pitch extraction, and melody matching. Onset detection captures the change of a certain feature in the audio signal by building a mathematical model, so as to detect the starting point of a sound. The specific methods include magnitude method (characterized by volume), short-term energy method (characterized by energy), and surf method (characterized by slope). The pitch extraction part estimates the fundamental frequency of each tone through autocorrelation function, average amplitude difference function, harmonic product spectrum, etc. Melody matching converts the extracted sequence into MIDI numbers and compares it with the digital sequence in the database. Common methods include hidden Markov model, dynamic programming, linear scaling, etc.

2.1.3 Advantages and Limitations of Content-Based Recommendation

Advantages of Content-Based Recommendation

(1) **No dependencies among users**. The construction of each user feature only depends on its own preferences for items (movies, books, music, etc.) and has nothing to do with the behavior of others. In contrast,

collaborative filtering algorithms need to use other user's interests to predict the target user's interests. This independence among users makes it less impactful on the recommendation performance even if the number of users in the recommender system is small.

(2) **Easy to explain**. In some specific scenarios, the recommender system needs to explain the reason to the user for recommending an item. It only needs to tell the user that the recommended item has certain attributes and these attributes often appear in the items that the user likes, to achieve the explanation of recommendation results.

(3) **Not restricted by new users or new items**. When a new user enters the recommender system, content-based recommendations can be made based on the user's personal attribute information, such as gender, age, occupation, and IP address. In the same way, new items can be recommended to users immediately after entering the recommender system. In the collaborative filtering algorithms, only the items evaluated by other users may be recommended to the target user. That is, the collaborative filtering algorithm is difficult to solve the cold-start problem, and the content-based recommendation algorithm will not suffer from the cold-start problem.

Limitations of Content-Based Recommendation

(1) **Difficulty of feature extraction**. If the item description in a recommender system is unstructured (such as books, movies, music, etc.), although the existing technology can extract some features to characterize the items, it is difficult to extract item features accurately and comprehensively. The problem that may be caused by incomplete feature extraction is that some features extracted from two items are very similar, but there may be huge differences in user preferences for them. For instance, movies with the same actor, director, and genre may have very different user ratings. In this case, a content-based recommendation algorithm cannot accurately distinguish the two movies.

(2) **Difficulty of discovering other potential interests of users, i.e., lack of diversity**. Content-based recommendation only depends on the

user’s personal attributes and historical preferences on contents, so the generated recommendation results will have a very high similarity to the user’s historical interactions. For instance, if a user has watched a lot of comedy movies in the past, a content-based recommendation system will only recommend comedy movies for him in most cases, without exploring whether he might like other types of movies.

These two limitations are exactly what the collaborative filtering algorithms are good at, so in the practical recommender systems, it is necessary to combine the collaborative filtering algorithm with the content-based recommendation algorithm.

2.2 Collaborative Filtering-Based Recommendation Algorithms

2.2.1 Memory-Based Collaborative Filtering

There is a popular proverb: birds of a feather flock together. It is a metaphor that items with similar characteristics are often placed in one place, and people with similar preferences are often gathered together. This proverb perfectly reveals the principle of memory-based recommendation algorithms. Memory-based recommendation algorithms calculate the similarity between users or items to generate their corresponding “clusters” for each target user, usually called neighbors. Since the “clusters” gather users who have similar preferences or characteristics with the target users, and similar users will have a large overlap in interacted items, the recommendation algorithm can calculate the preferences of the target users based on the interaction records of other users in the same “clusters,” so as to generate a recommendation list for each of them.

This section first introduces two classic memory-based collaborative filtering algorithms: user-based collaborative filtering algorithm and item-based collaborative filtering algorithm, which generate recommendation lists for users from two different perspectives of user similarity and item similarity. Then, we introduce improved memory-based collaborative filtering algorithms such as SLIM, SSLIM, and LorSLIM. The principle of these algorithms is the same as the classic memory-based collaborative filtering method, but their accuracy is greatly improved.

Classic Memory-Based Collaborative Filtering Algorithms

Generally speaking, users in a recommender system are not independent from each other, and there will be certain similarities between some users, for instance, they like the same music and singers. These similarities will implicitly associate two users, so that the interests and preferences of one user will be implicitly influenced by another user. This implicit association is often referred to as "implicit relationship." If two users have higher similarities, then the "implicit relationship" between the two users will be stronger, and the influence of interests and preferences between the two users will be more significant. For instance, users *A*, *B*, *C* interact with items *a*, *b*, *c*, *d*, *e*, and the user-item interaction relationship is shown as Fig. 2.10.

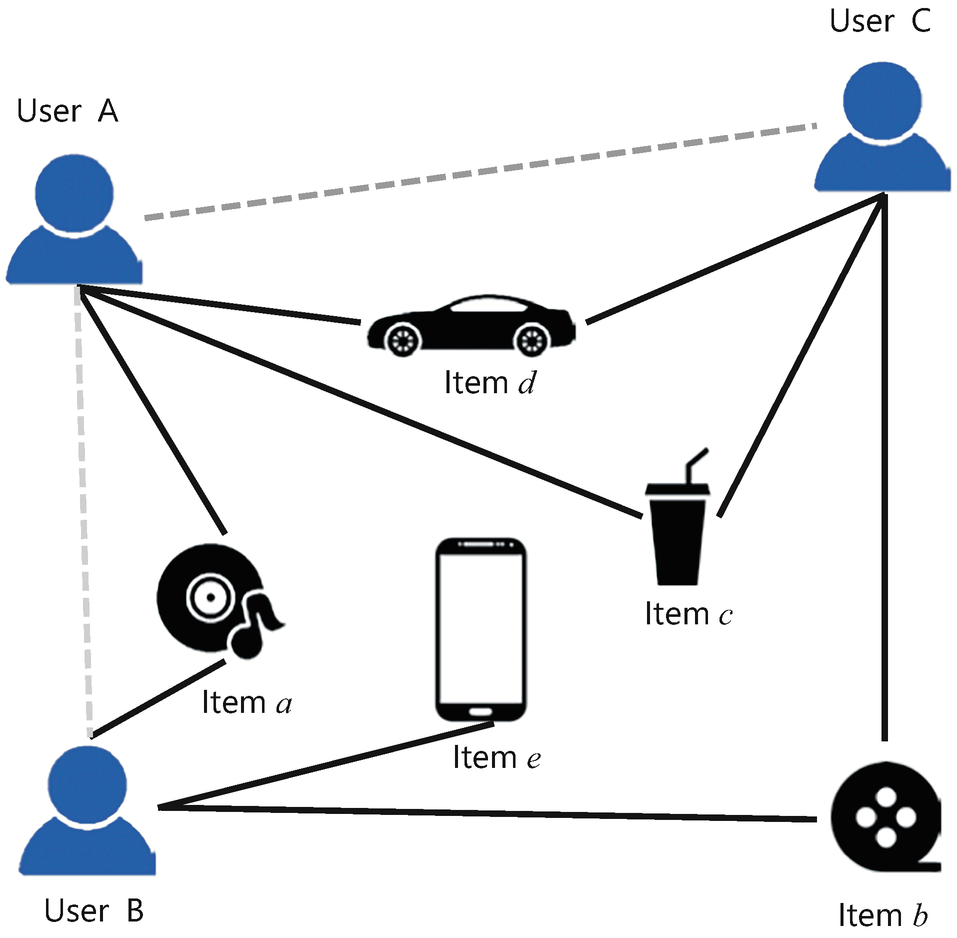


Fig. 2.10 Example of user-item interaction (user-centric view)

The solid line indicates the interaction between users and items, the dotted line indicates the hidden relationship between users, and the color of the dotted line indicates the strength of the hidden relationship, that is, the darker the dotted line, the stronger the hidden relationship. In Fig. 2.10, users *A* and *B* interact with item *a*, so it can be considered that there is a certain similarity between the two users. Similarly, users *A* and *C* jointly interact with items *c* and *d*, then there is a certain similarity between these two users. These similarities between users can be expressed by the implicit relationship mentioned earlier, since the number of co-interacted items between users *A* and *C* is more than that between users *A* and *B*, so the implicit relationship between user *A* and user *C* is stronger than the implicit relationship between user *A* and user *B*, and the influence between user *A* and user *C* on interests and preferences is more significant than those between user *A* and user *B*. Therefore, in the recommendation process, we are more likely to recommend the item *b* that user *C* has interacted with to user *A* than the item *e* that user *B* has interacted with.

User-based collaborative filtering algorithm [6] first mines the hidden relationship between users according to the user's interaction records and builds a set of users similar to the target user, also known as a neighbor set. Then a corresponding recommendation list is generated based on the interests and preferences of the neighbors. Generally speaking, the recommendation process of the algorithm can be divided into two stages: similar user set calculation and recommendation list generation.

The similar user set calculation phase mainly calculates the similarity between users based on the existing user interaction records and then filters users with high similarity with the target user according to the similarity and the preset threshold to generate a similar user set. Among them, cosine similarity, Jaccard similarity, Pearson coefficient, Euclidean distance, Manhattan distance, etc. can be used to calculate the similarity or distance between users. Here, we mainly introduce two commonly used methods—Jaccard similarity coefficient and cosine similarity.

The Jaccard similarity is specially used to calculate the similarity between finite sets. Generally speaking, the larger the Jaccard similarity of two sets, the higher the similarity between the two sets. The Jaccard similarity is calculated as follows:

$$S_{u,v} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}, \tag{2.18}$$

where $S_{u,v}$ represents the similarity between user *u* and user *v*; $N(u)$ represents the set of items user *u* has interacted with; $N(v)$ represents the set of items that user *v* has interacted with.

Cosine similarity measures the similarity of two vectors according to the angle between two vectors in the coordinate system. The smaller the angle between two vectors, the more similar the two vectors are, and vice versa. Its calculation is described as follows:

$$S_{u,v} = \frac{V(u) \cdot V(v)}{\sqrt{|V(u)||V(v)|}}, \tag{2.19}$$

where $S_{u,v}$ represents the similarity between user *u* and user *v*; $V(u)$ represents the rating vector of user *u* on items; $V(v)$ represents the rating vector of user *v* on items. It is worth noting that the above cosine similarity calculation formula does not consider user rating scale, that is, different users may give the same item different ratings according to their preferences or habits. Therefore, the above calculation formula needs to be adjusted to take into account user rating preferences or habits. The adjusted cosine similarity calculation process is as follows:

$$S_{u,v} = \frac{\sum_{c \in I_{u,v}} (R_{u,c} - \bar{R}_u)(R_{v,c} - \bar{R}_v)}{\sqrt{\sum_{c \in I_u} (R_{u,c} - \bar{R}_u)^2} \sqrt{\sum_{c \in I_v} (R_{v,c} - \bar{R}_v)^2}}, \tag{2.20}$$

where $I_{u,v}$ represents the set of items that user *u* and user *v* have interacted with; $R_{u,c}$ represents the rating of user *u* on item *c*; $R_{v,c}$ represents the rating of user *v* on item *c*; $\bar{R}_u$ represents the average rating of user *u* on all interacted items; $\bar{R}_v$ represents the average rating of user *v* on all interacted items; I_u represents the set of items that user *u* has interacted with; I_v represents the set of items user *v* has interacted with.

After using the above methods to calculate the similarity between any two users, the similarity matrix $\boldsymbol{X}_{n \times n}$ among all users can be obtained. Among them, *n* represents the number of users, and each element of matrix $X_{i,j}$ represents the similarity between user *i* and user *j*. Assuming that we

want to find the *K* users who are most similar to the target user, we can find the top *K* users with the largest similarity value from the corresponding row of the similarity matrix $\boldsymbol{X}$ to form a similar user set, which will be used in the next stage to generate a list of recommendations for each user.

In the recommendation list generation stage, a corresponding recommendation list is generated for the target user based on the set of similar users. To generate a recommendation list for each user in the system, it is first necessary to calculate each user's preference for the candidate items as follows:

$$\tilde{R}_{u,i} = \frac{\sum_{v \in S(u,K)} S_{u,v} \times R_{v,i}}{\sum_{v \in S(u,K)} |S_{u,v}|}, \tag{2.21}$$

where $\tilde{R}_{u,i}$ represents the predicted rating of user *u* on item *i*; $R_{v,i}$ represents the rating of user *v* on item *i*; $S(u, K)$ represents the set of *K* users who are most similar to user *u*. Then, all candidate items are sorted in descending order according to the predicted ratings, and the top *N* (for example, the top 5 or top 10) items are selected to form the user's recommendation list. Finally, we can recommend the list of items to the target user.

The item-based collaborative filtering algorithm [16] completes the recommendations to users from the perspective of item similarity. The similarity of the candidate item should be positively correlated with the items that the target user interacted with before, that is, the more similar the item is to the item that the user has interacted with before, the more likely the item will be the user's next interaction. Therefore, item-based collaborative filtering first calculates the similarity between items based on user interaction records, thereby mining the "hidden relationship" between items. According to the implicit relationship between items, we can calculate the possibility of the candidate items interacting with the user in the future, and generate the user's recommendation list accordingly. For instance, items *a*, *b*, *c*, *d*, *e* have interactions with users *A*, *B*, and *C*, and the interaction relationship is shown in Fig. 2.11. Items that user *A* has not interacted with are *b* and *e*. For item *b*, since there is a common interacted user between it and other items, it has the same similarity with items *a*, *c*, and *d*, showing item *b* and *a*, *c*, and *d* have the same strength of implicit relations. For item *e*, it only has a certain degree of similarity with item *a*,

and the similarity with other items is 0, showing that there is a certain implicit relationship between item *e* and item *a* and there is no implicit relationship between *e* and other items. Therefore, when recommending items to user *A*, the algorithm will be more likely to recommend item *b* rather than item *e*, because item *b* has greater similarity with items that user *A* interacted with before, i.e., the implicit relationship is stronger.

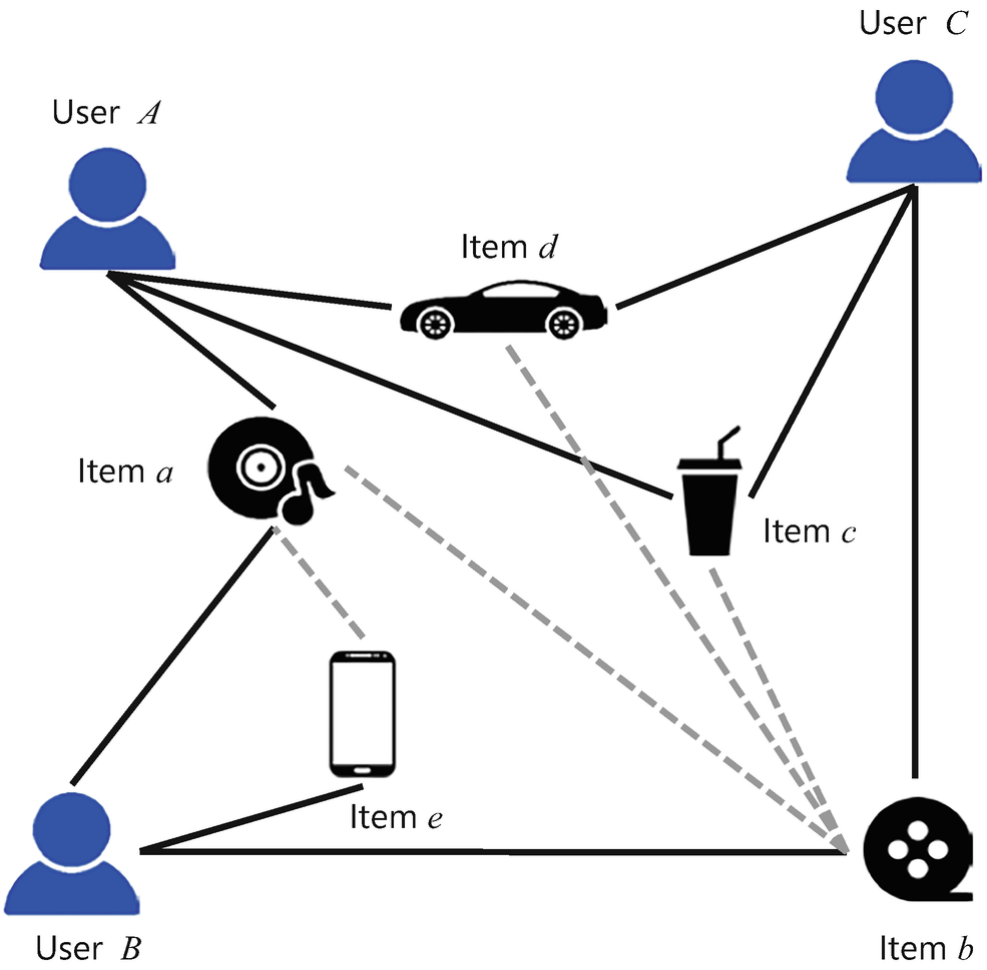


Fig. 2.11 Example of user-item interaction (item-centric view)

The computation process of the item-based collaborative filtering algorithm can also be divided into two stages: similar item set calculation and user recommendation list generation. The similar item set calculation stage calculates the similarity between each pair of items according to the user interaction records, so as to obtain the top *K* items most similar to each target item to form the similar item set. First, it is necessary to calculate the similarity between each item and all other items and construct an item--item similarity matrix. Then, we can filter out *K* items with the largest similarity

from the corresponding rows of the items in the similarity matrix to form a set of similar items and generate a user recommendation list based on this set. The method for calculating the similarity between items is the same as the method for calculating the similarity between users and will not be repeated here.

In the recommendation list generation stage, we first calculate the user's predicted score for each candidate item, then sort all items according to the predicted score, and finally recommend *N* items with the highest scores to the user. Similar to the user-based collaborative filtering algorithm, assuming that the set of *K* most similar items of the target item *i* is $S(i, K)$, the predicted score of the user *u* on the item *i* can be calculated by the following formula:

$$\tilde{R}_{u,i} = \frac{\sum_{j \in S(i,K)} S_{i,j} \cdot R_{u,j}}{\sum_{j \in S(i,K)} |S_{i,j}|}, \tag{2.22}$$

where $\tilde{R}_{u,i}$ represents the predicted rating of user *u* on item *i*; $S_{i,j}$ represents the similarity between item *i* and item *j*; $R_{u,j}$ represents user *u*'s rating on item *j*. It can be seen from the formula that the predicted score of user *u* for recommended item *i* is equal to the weighted sum of scores of items that user *u* has interacted with and are similar to the target item.

In general, the user-based collaborative filtering algorithm starts from the perspective of the target user and selects items that are liked by users who are highly similar to the target user as the recommendation results. The item-based collaborative filtering recommendation algorithm starts from the perspective of candidate items and selects items that are similar to the items that the target user has interacted with as the recommendation results. These two algorithms are simple and easy to understand in principle, clear and complete in structure, and easy to use and practice.

Based on the above works, many researchers proposed several improvement methods from the perspective of recommendation accuracy, and the main improvement ideas are summarized as follows:

(1) **Choose an appropriate similarity calculation method**. For the same dataset, if different similarity calculation methods are used, the accuracy of the recommendation will also be significantly different. For instance, using the Pearson correlation coefficient as a similarity calculation method needs to meet two assumptions, that is, the

relationship between variables is linear, and the error needs to satisfy a probability distribution with a mean of 0 and a constant variance. If the dataset does not meet the above conditions, there will be a large deviation in the recommendation results. Therefore, it is necessary to select an appropriate similarity calculation method according to the characteristics of the dataset distribution and research questions.

(2) **The reliability of similarity calculation**. If there are a large number of commonly interacted items between a user and the target user, the user should be given a larger weight when calculating the prediction scores of items, that is, the similarity between the two users is more reliable. On the contrary, it means that the reliability of the similarity is lower. Usually, we can set a threshold for the number of commonly interacted items, such as 50. When the number of common interactions is less than the threshold, the user can be given a weight less than 1 when calculating the prediction score, otherwise the user is given a weight of 1, that is, the neighbor's score should punished according to the reliability.

(3) **Choose an appropriate number of neighbors to compute the prediction score**. The number of users in recommender systems is usually large, so it is unrealistic to consider all users as neighbors when calculating the score. It is necessary to select some users as neighbors of the target user according to certain rules (for example, the similarity is higher than a certain threshold). The specific number of neighbors is related to the dataset and needs to be set by manual tuning.

(4) **Consider user rating habits to avoid rating prediction bias caused by different user rating habits**. In the real world, each user has her/his own rating habits. For instance, some users are used to giving high ratings to movies, while others are used to giving low ratings to movies. Therefore, it is necessary to eliminate the prediction bias caused by different user rating preferences. A feasible method is to subtract the average value of the corresponding user's rating from the rating of each item before calculating the prediction score and then add the average value back after the rating prediction stage, so as to avoid the influence of different users' rating preferences.

Advanced Memory-Based Collaborative Filtering Algorithms

The SLIM (Sparse Linear Methods) [10] algorithm proposes some improvements to memory-based methods. The SLIM algorithm proposes to learn a sparse matrix for all items from user interaction records to simultaneously achieve efficient and high-quality recommendations. Specifically, if we want to calculate the rating of user *i* on an item *j* that has not yet been interacted with, it can be achieved by performing sparse aggregation on the ratings of items that user *i* has interacted with, namely

$$\tilde{a}_{ij} = \boldsymbol{a}_i^\top \boldsymbol{w}_j, \tag{2.23}$$

where $a_{ij} = 0$ and $\boldsymbol{w}_j \in \mathbb{R}^n$ is a sparse column vector. From the perspective of matrix operations, the SLIM model can be expressed as

$$\tilde{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{W}, \tag{2.24}$$

where $\boldsymbol{A}$ represents the user interaction matrix; $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ represents a sparse matrix; its *j*-th column is the $\boldsymbol{w}_j$ in the above formula; each row $\tilde{\boldsymbol{a}}_i^\top$ in $\tilde{\boldsymbol{A}}$ matrix represents the predicted ratings of user *i* on all items.

The focus of SLIM is the construction of sparse matrix $\boldsymbol{W}$, which can be obtained through the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \frac{1}{2}\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{W}\|_{\mathrm{F}}^2 + \frac{\beta}{2}\|\boldsymbol{W}\|_{\mathrm{F}}^2 + \lambda\|\boldsymbol{W}\|_1. \\ \text{s.t.} \quad & \boldsymbol{W} \geq 0, \\ & \operatorname{diag}(\boldsymbol{W}) = 0. \end{aligned} \tag{2.25}$$

In the above problem, $\|\boldsymbol{W}\|_1 = \sum_{i=1}^n \sum_{j=1}^n |\boldsymbol{w}_{ij}|$ represents the ℓ_1 norm of the sparse matrix $\boldsymbol{W}$; $\|\boldsymbol{W}\|_{\mathrm{F}}^2$ represents the ℓ_{F} norm of the sparse matrix $\boldsymbol{W}$; β and λ represent the coefficients of the regularization terms, and the larger the coefficient of the regularization term, the stricter the constraint on the parameters. In the optimization objective, the first term $\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{W}\|_{\mathrm{F}}^2$ describes the fitting error between the predicted value and the real value, and both the second term and the third term are regularization terms which are used to penalize the value of the sparse matrix $\boldsymbol{W}$. The introduction of the third term ℓ_1 norm makes the matrix $\boldsymbol{W}$ tend to be sparse (that is, multiple elements in the matrix are 0). The

introduction of the second term ℓ_{F} norm can transform the optimization problem into an elastic net regression problem, which is used to reduce the complexity of the model and avoid overfitting. In addition, the first constraint $\boldsymbol{W} \geq 0$ ensures that each item in the sparse matrix is positively correlated, and the second constraint ensures that the sparse matrix $\boldsymbol{W}$ is not a trivial solution, that is, $\boldsymbol{W}$ is not an identity matrix and $\tilde{a}_{ij}$ does not rely on a_{ij} when calculating. Since each column of the matrix $\boldsymbol{W}$ is independent, the construction of $\boldsymbol{W}$ is highly parallelizable. SLIM can also reduce training time by combining with feature selection methods. For instance, SLIM combined with feature selection methods such as cosine similarity can greatly improve training efficiency with a slight decrease in recommendation quality.

Compared with traditional linear models, SLIM has significant advantages. For instance, the linear model based on the item-based k-nearest neighbor (ItemKNN) algorithm is similar to SLIM in principle. It uses an item--item cosine similarity matrix $\boldsymbol{S} \in \mathbb{R}^{m \times k}$ to achieve item recommendation, but ItemKNN relies too much on the pre-calculated item--item similarity matrix $\boldsymbol{S}$, while SLIM obtains the similarity matrix $\boldsymbol{W}$ by an optimization problem, so that $\boldsymbol{W}$ can encode the relationship information between items that is not easy to be captured by the similarity calculation method. In addition, $\boldsymbol{S}$ is a dense symmetric matrix, and the value of the matrix can be negative. While the similarity matrix $\boldsymbol{W}$ obtained by SLIM optimization is a highly sparse non-negative matrix, making SLIM has extremely high recommendation efficiency. At the same time, since $\boldsymbol{W}$ is not required to be a symmetric matrix, SLIM has better flexibility.

SLIM and Matrix Factorization (MF) methods are also quite similar in structure. Matrix factorization reconstructs the user-item interaction matrix $\tilde{\boldsymbol{A}}$ through the user feature matrix $\boldsymbol{U}$ and item feature matrix $\boldsymbol{V}$, the specific calculation is as follows:

$$\tilde{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{V}^{\top}. \tag{2.26}$$

It can be seen from the above equation that SLIM is essentially a special form of matrix factorization, that is, $\boldsymbol{A}$ and $\boldsymbol{W}$ of SLIM can correspond to $\boldsymbol{U}$ and $\boldsymbol{V}^{\top}$ of matrix factorization, respectively. Matrix factorization needs to construct respective feature matrices $\boldsymbol{U}$ and $\boldsymbol{V}^{\top}$ for users and items, while SLIM only needs to build feature matrices $\boldsymbol{W}$ for items, so the

learning process of SLIM is simpler than matrix factorization. In addition, $\boldsymbol{U}$ and $\boldsymbol{V}^\top$ are usually constructed as low-dimensional hidden spaces, resulting in the possible loss of some useful high-dimensional user features and item features when decomposing $\boldsymbol{A}$ into $\boldsymbol{U}$ and $\boldsymbol{V}^\top$. On the contrary, user information in SLIM is completely retained in the user interaction matrix $\boldsymbol{A}$, so SLIM is better than matrix factorization in terms of recommendation accuracy in many Top-*N* recommendation scenarios. In terms of recommendation efficiency, SLIM is also better than matrix factorization. Since $\boldsymbol{U}$ and $\boldsymbol{V}^\top$ are both dense matrices in matrix decomposition, when calculating the predicted score vector $\tilde{\boldsymbol{a}}_i$ of user *i*, it is necessary to calculate the score $\tilde{a}_{ij}$ of each item separately, and its time complexity is $O(k^2 \times n)$, where *k* represents the dimensions of $\boldsymbol{U}$ and $\boldsymbol{V}^\top$, *n* represents the number of items. The SLIM method using sparse matrix can reduce the time complexity of the algorithm and improve the training efficiency.

At present, many related research works have proposed some improvement ideas to optimize the SLIM algorithm. For instance, SSLIM [11] introduces side information based on SLIM to improve the accuracy of recommendation. SSLIM proposes two methods to use auxiliary information. The first method is to share a sparse matrix $\boldsymbol{W}$ with the user interaction matrix $\boldsymbol{A}$ to reconstruct the auxiliary information matrix $\boldsymbol{F}$, whose optimization objective is:

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \frac{1}{2}\|\boldsymbol{A}-\boldsymbol{A}\boldsymbol{W}\|_{\mathrm{F}}^2+\frac{\alpha}{2}\|\boldsymbol{F}-\boldsymbol{F}\boldsymbol{W}\|_{\mathrm{F}}^2+\frac{\beta}{2}\|\boldsymbol{W}\|_{\mathrm{F}}^2+\lambda\|\boldsymbol{W}\|_1 \\ \text{s.t.} \quad & \boldsymbol{W}\geq 0, \\ & \operatorname{diag}(\boldsymbol{W})=0. \end{aligned} \tag{2.27}$$

In the formula, α is a regularization coefficient, which controls the importance of auxiliary information in the training phase. The larger value of α means the auxiliary information is more important for model training. The second method is to set a separate sparse matrix $\boldsymbol{Q}$ for the auxiliary information matrix $\boldsymbol{F}$, but it is necessary to ensure that $\boldsymbol{W}$ and $\boldsymbol{Q}$ are as close as possible:

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{Q}} \quad & \frac{1}{2}\|\boldsymbol{A}-\boldsymbol{A}\boldsymbol{W}\|_{\mathrm{F}}^2+\frac{\alpha}{2}\|\boldsymbol{F}-\boldsymbol{F}\boldsymbol{Q}\|_{\mathrm{F}}^2+\frac{\beta}{2}\|\boldsymbol{W}-\boldsymbol{Q}\|_{\mathrm{F}}^2+\lambda(\|\boldsymbol{W}\|_1+\|\boldsymbol{Q}\|_1) \\ \text{s.t.} \quad & \boldsymbol{W}\geq 0, \boldsymbol{Q}\geq 0, \\ & \operatorname{diag}(\boldsymbol{W})=0, \operatorname{diag}(\boldsymbol{Q})=0. \end{aligned} \tag{2.28}$$

LorSLIM [3] uses a kernel norm on top of SLIM to ensure the low-rank characteristic of the sparse matrix. The low-rank characteristic of the sparse matrix can enable LorSLIM to better capture the relationship between items on sparse data. At the same time, the low rank and sparsity of the sparse matrix ensure that it is a block diagonal matrix, and similar items will be classified into the same category. LorSLIM obtains a low-rank sparse matrix $\boldsymbol{W}$ by optimizing the following objective function:

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \frac{1}{2}\|\boldsymbol{A}-\boldsymbol{A}\boldsymbol{W}\|_{\mathrm{F}}^2+\frac{\beta}{2}\|\boldsymbol{W}\|_{\mathrm{F}}^2+\lambda\|\boldsymbol{W}\|_1+z\|\boldsymbol{W}\|_* \\ \text{s.t.} \quad & \boldsymbol{W}\geq 0, \\ & \operatorname{diag}(\boldsymbol{W})=0. \end{aligned} \tag{2.29}$$

In the formula, $\|\boldsymbol{W}\|_*=\sum_{i=1}^{\operatorname{rank}(\boldsymbol{W})}\sigma_i$ represents the nuclear norm of $\boldsymbol{W}$; σ_i is the singular value of the matrix $\boldsymbol{W}$; z represents the coefficient of the regularization term. However, due to the introduction of the nuclear norm, LorSLIM cannot use coordinate descent and soft thresholding methods to obtain $\boldsymbol{W}$. Therefore, LorSLIM uses ADMM (Alternating Direction Method of Multipliers) to solve the optimization problem. Readers who are interested in solving optimization problems with ADMM can learn more by reading related papers.

Summary of Memory-Based Collaborative Filtering Algorithms

The user-based collaborative filtering method and the item-based collaborative filtering method are two classic memory-based methods, which calculate the user recommendation lists from the perspective of user similarity and item similarity, respectively. They are simple and easy to understand in principle, clear and complete in structure, easy for readers and beginners to get started and practice. The SLIM algorithm and its variants SSLIM algorithm and LorSLIM algorithm are improvements to memory-based methods. They generate user recommendation lists from the perspective of item similarity. Compared with memory-based methods, the

similarity matrices of these methods are optimized and sparse, which can achieve both fast and high-quality recommendations. The LorSLIM algorithm has a better performance on sparse datasets due to the introduction of the nuclear norm. However, both the memory-based method and the SLIM algorithm or its variants generally have a shortcoming, that is, they cannot solve the recommendation problem of new users and new items. Recommender systems based on the above methods cannot generate recommendation lists for new users and cannot recommend new items to other users, which brings great challenges to the application of recommender systems.

2.2.2 Matrix Factorization Method and Factorization Machine Method

The matrix factorization method was introduced into the collaborative filtering algorithm to solve the data sparsity [13] problem. In the Netflix Prize competition held in 2006, the matrix factorization method achieved the highest accuracy among all stand-alone CF algorithms and has since attracted widespread attention from academia and industry. In 2010, Rendle extended the matrix factorization method and proposed the factorization machine method [13], which can model more complex relationships between users and items.

Matrix Factorization Method

(1) Classic Matrix Factorization

As shown in Fig. 2.12, the classic Matrix Factorization (MF) method is a simple embedding model that can be used in recommender systems. Its core idea is to find a low-dimensional space to represent users and items. Specifically, given a user-item interaction matrix $\boldsymbol{R} \in \mathbb{R}^{m \times n}$, *m* represents the total number of users, *n* represents the total number of items, the matrix factorization will learn:

- User embedding matrix $\boldsymbol{U} \in \mathbb{R}^{m \times d}$, where the *i*-th row represents the embedding of user *i*.

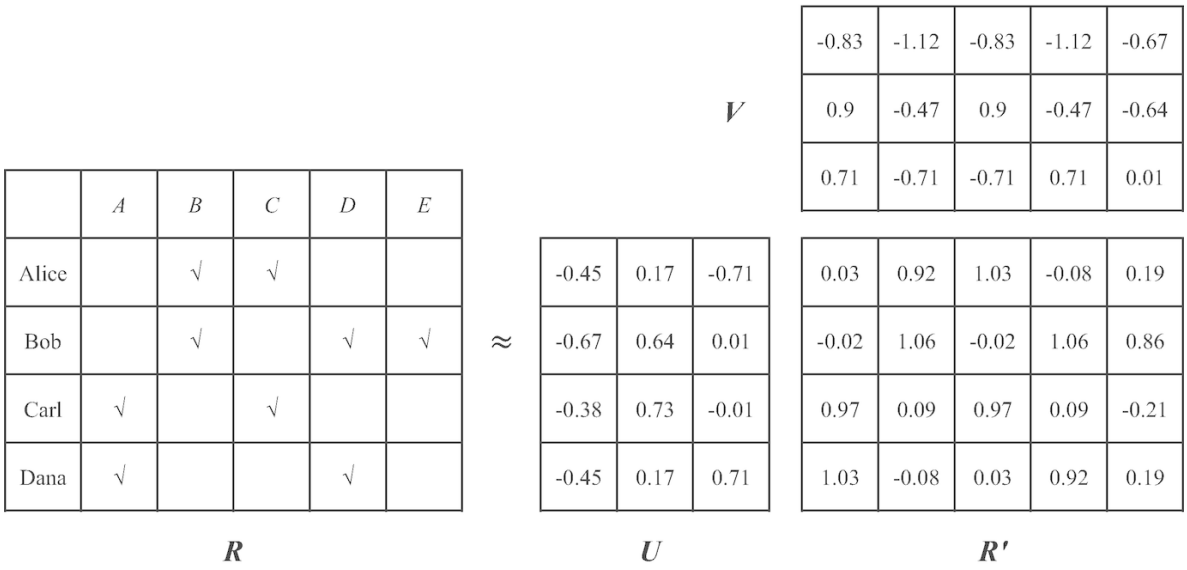


Fig. 2.12 Illustration of matrix factorization

- Item embedding matrix $\boldsymbol{V} \in \mathbb{R}^{n \times d}$, where the *j*-th row represents the embedding of item *j*.

Embeddings are learned such that the product $\boldsymbol{R}' = \boldsymbol{U}\boldsymbol{V}^\top$ is a reasonable approximation of the interaction matrix $\boldsymbol{R}$, where the entry (i, j) in R' is the product of the representation vectors of user *i* and item *j*. In order to make R'_{ij} as close as possible to $R_{i,j}$, the optimization objective $\mathcal{L}$ of matrix factorization can be defined as follows:

$$\mathcal{L} = \sum_{i=1}^{m} \sum_{j=1}^{n} (R_{ij} - \boldsymbol{U}_i \boldsymbol{V}_j^\top)^2. \tag{2.30}$$

Meanwhile, in order to ensure that the matrix factorization model does not suffer from the overfitting problem, it is necessary to add L_2 regularization to the objective function $\mathcal{L}$, namely:

$$\mathcal{L} = \sum_{i=1}^{m} \sum_{j=1}^{n} (R_{ij} - \boldsymbol{U}_i \boldsymbol{V}_j^\top)^2 + \lambda(\|\boldsymbol{U}\|^2 + \|\boldsymbol{V}\|^2). \tag{2.31}$$

To solve the above optimization problem, the gradient descent method is usually used, such as the stochastic gradient descent (SGD) method, and the partial derivative of the objective function $\mathcal{L}$ to $\boldsymbol{U}, \boldsymbol{V}$ is calculated as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}} = -2\boldsymbol{R}\boldsymbol{V} + 2\lambda \boldsymbol{U}, \tag{2.32}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}} = -2\boldsymbol{R}^{\top}\boldsymbol{U} + 2\lambda \boldsymbol{V}. \tag{2.33}$$

Based on the above partial derivatives, the iterative update formula of SGD is obtained by:

$$\boldsymbol{U} = \boldsymbol{U} - \alpha * \frac{\partial \mathcal{L}}{\partial \boldsymbol{U}}, \tag{2.34}$$

$$\boldsymbol{V} = \boldsymbol{V} - \alpha * \frac{\partial \mathcal{L}}{\partial \boldsymbol{V}}. \tag{2.35}$$

By continuously iterating using the above formulas, the $\boldsymbol{U}$ and $\boldsymbol{V}$ obtained after convergence can be used as the recommendation model to recommend items for users.

(2) Probabilistic Matrix Factorization

Although the classic matrix factorization method introduced above has excellent results in practical applications, there are some key technical problems that have not yet been resolved, such as whether the mean square error is reasonable, and how to select the coefficient of the regularization term, etc. In order to solve the above problems, Ruslan Salakhutdinov and Andriy Mnih proposed the idea of Probabilistic Matrix Factorization (PMF) [15]. Different from the classical matrix factorization, the probabilistic matrix factorization assumes that the observed rating matrix has noises, and the noises follow the Gaussian distribution with zero mean, and the user feature vector and the item feature vector also follow the Gaussian distribution. Based on these assumptions, probabilistic matrix factorization can solve the above problems.

As shown in Fig. 2.13, in the probabilistic graphical model of probabilistic matrix factorization, the user feature vector and the item feature vector are initialized using an isotropic multivariate Gaussian with zero mean. Suppose there are *m* users and *n* items in the recommender system, and the dimension of user feature vector and the item feature vector is *d*, then we have:

- User embedding matrix $\boldsymbol{U} \in \mathbb{R}^{m \times d}$, where the *i*-th row represents the embedding of user *i*.

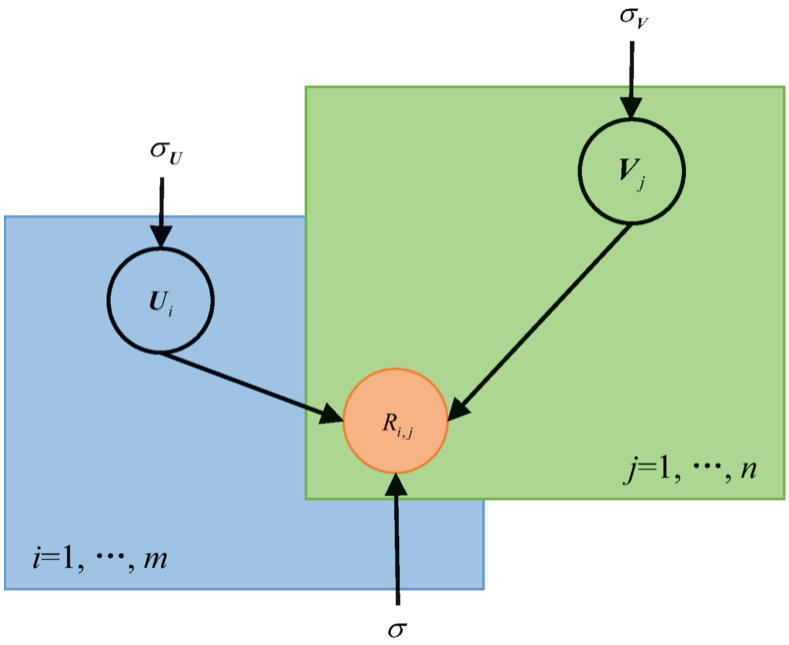


Fig. 2.13 Probabilistic graphical model of probabilistic matrix factorization

- Item embedding matrix $\boldsymbol{V} \in \mathbb{R}^{n \times d}$, where the *j*-th row represents the embedding of item *j*.

The core of the probabilistic matrix factorization is Bayesian theory, and Bayes' theorem can be used to estimate the posterior distribution of model parameters, as follows:

$$p(\theta \mid \boldsymbol{R}, \alpha) = \frac{p(\boldsymbol{R} \mid \theta, \alpha) p(\theta \mid \alpha)}{p(\boldsymbol{R} \mid \alpha)} \propto p(\boldsymbol{R} \mid \theta, \alpha) p(\theta \mid \alpha), \tag{2.36}$$

where $\boldsymbol{R}$ is the user's rating matrix for items; θ is the parameter set of the distribution; α is the hyperparameter of the distribution; $p(\theta \mid \boldsymbol{R}, \alpha)$ is the posterior distribution of θ; $p(\theta \mid \alpha)$ is the prior distribution; $p(\boldsymbol{R} \mid \theta, \alpha)$ is the likelihood function. The idea to solve the above problem is: as more information about the data distribution is obtained, the model adjusts the parameters θ to fit the data. That is, the parameters of the posterior distribution of the previous iteration are used as the prior distribution for the next iteration until the posterior distribution $p(\theta \mid \boldsymbol{R}, \alpha)$ tends to be stable.

For ease of notation, we can define $\theta = \{\boldsymbol{U}, \boldsymbol{V}\}, \alpha = \sigma^2$, where σ represents the standard deviation of the zero-mean Gaussian distribution. Using the above definition to rewrite the above formulas, we can obtain:

$$p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \sigma^2) = p(\boldsymbol{R} \mid \boldsymbol{U}, \boldsymbol{V}, \sigma^2) p(\boldsymbol{U}, \boldsymbol{V} \mid \sigma_{\boldsymbol{U}}^2, \sigma_{\boldsymbol{V}}^2). \tag{2.37}$$

Since $\boldsymbol{U}$ and $\boldsymbol{V}$ are independent of each other, the above formula can be rewritten as:

$$p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \sigma^2) = p(\boldsymbol{R} \mid \boldsymbol{U}, \boldsymbol{V}, \sigma^2) p(\boldsymbol{U} \mid \sigma_{\boldsymbol{U}}^2) p(\boldsymbol{V} \mid \sigma_{\boldsymbol{V}}^2). \tag{2.38}$$

In the above equation, $p(\boldsymbol{R} \mid \boldsymbol{U}, \boldsymbol{V}, \sigma^2)$ is the likelihood function, defined as follows:

$$p(\boldsymbol{R} \mid \boldsymbol{U}, \boldsymbol{V}, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} [\mathcal{N}(\boldsymbol{R}_{ij} \mid \boldsymbol{U}_i^\top \boldsymbol{V}_j, \sigma^2)]^{I_{ij}}, \tag{2.39}$$

where I_{ij} is an indicator function, that is, if $\boldsymbol{R}_{ij}! = 0$, then I_{ij} is 1, otherwise it is 0; $\boldsymbol{U}_i^\top \boldsymbol{V}_j$ is the mean of Gaussian distribution; σ^2 is the variance of Gaussian distribution.

In addition, $p(\boldsymbol{U} \mid \sigma_{\boldsymbol{U}}^2)$ and $p(\boldsymbol{V} \mid \sigma_{\boldsymbol{V}}^2)$ are the prior distribution, defined as:

$$p(\boldsymbol{U} \mid \sigma_{\boldsymbol{U}}^2) = \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{U}_i \mid 0, \sigma_{\boldsymbol{U}}^2), \tag{2.40}$$

$$p(\boldsymbol{V} \mid \sigma_{\boldsymbol{V}}^2) = \prod_{j=1}^{M} \mathcal{N}(\boldsymbol{V}_j \mid 0, \sigma_{\boldsymbol{V}}^2). \tag{2.41}$$

That is, two zero-mean Gaussian distributions. Combining the above definitions, we can get:

$$\begin{aligned} p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \sigma^2) = & \prod_{i=1}^{N} \prod_{j=1}^{M} [\mathcal{N}(\boldsymbol{R}_{ij} \mid \boldsymbol{U}_i^\top \boldsymbol{V}_j, \sigma^2)]^{I_{ij}} \\ & \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{U}_i \mid 0, \sigma_{\boldsymbol{U}}^2) \prod_{j=1}^{M} \mathcal{N}(\boldsymbol{V}_j \mid 0, \sigma_{\boldsymbol{V}}^2). \end{aligned} \tag{2.42}$$

In order to solve the optimization problem of the probabilistic matrix factorization, it is necessary to maximize the above objective to obtain the optimal parameters $\boldsymbol{U}$, $\boldsymbol{V}$. It is difficult to obtain derivatives for the optimization objective in the form of multiplication, so it is difficult to directly solve $p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \sigma^2)$. Here, we can consider to take the logarithmic on both sides of the equation in the above formula and we can get:

$$\begin{aligned} \ln p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \sigma^2) = & \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} \ln[\mathcal{N}(\boldsymbol{R}_{ij} \mid \boldsymbol{U}_i^\top \boldsymbol{V}_j, \sigma^2)] \\ & + \sum_{i=1}^{N} \ln \mathcal{N}(\boldsymbol{U}_i \mid 0, \sigma_{\boldsymbol{U}}^2) + \sum_{j=1}^{M} \ln \mathcal{N}(\boldsymbol{V}_j \mid 0, \sigma_{\boldsymbol{V}}^2). \end{aligned} \tag{2.43}$$

The Gaussian probability density function in the above formula is defined as:

$$\mathcal{N}(X \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right). \tag{2.44}$$

Combining the above two equations, we can get:

$$\begin{aligned} \ln p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \sigma^2) = & -\frac{1}{2\sigma^2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} (\boldsymbol{R}_{ij} - \boldsymbol{U}_i^\top \boldsymbol{V}_j)^2 \\ & - \frac{1}{2\sigma_{\boldsymbol{U}}^2} \sum_{i=1}^{N} \boldsymbol{U}_i^2 - \frac{1}{2\sigma_{\boldsymbol{V}}^2} \sum_{j=1}^{M} \boldsymbol{V}_j^2. \end{aligned} \tag{2.45}$$

The above equation is convenient for derivation. Finally, in order to facilitate the processing of the equation, we can define the hyperparameter $\lambda_{\boldsymbol{U}} = \frac{\sigma_{\boldsymbol{U}}^2}{\sigma^2}$, $\lambda_{\boldsymbol{V}} = \frac{\sigma_{\boldsymbol{V}}^2}{\sigma^2}$ and extract the relevant common factors to obtain:

$$\mathcal{L} = -\frac{1}{2}\left(\sum_{i=1}^{N} \sum_{j=1}^{M} (\boldsymbol{R}_{ij} - \boldsymbol{U}_i^\top \boldsymbol{V}_j)^2_{(i,j) \in \Omega_{\boldsymbol{R}_{ij}}} + \lambda_{\boldsymbol{U}} \sum_{i=1}^{N} \boldsymbol{U}_i^2 + \lambda_{\boldsymbol{V}} \sum_{j=1}^{M} \boldsymbol{V}_j^2\right). \tag{2.46}$$

Find the partial derivatives of $\mathcal{L}$ to $\boldsymbol{U}_i, \boldsymbol{V}_j$, respectively, and let the partial derivative be 0, the following equation can be obtained:

$$\nabla_{\boldsymbol{U}_i}\mathcal{L} = \left[\sum_{j=1}^{M}(\boldsymbol{R}_{ij} - \boldsymbol{U}_i^\top \boldsymbol{V}_j)\boldsymbol{V}_j^\top\right]_{(i,j)\in\Omega_{\boldsymbol{R}_{ij}}} - \lambda_{\boldsymbol{U}}\boldsymbol{U}_i = 0, \tag{2.47}$$

$$\nabla_{\boldsymbol{V}_j}\mathcal{L} = \left[\sum_{i=1}^{N}(\boldsymbol{R}_{ij} - \boldsymbol{U}_i^\top \boldsymbol{V}_j)\boldsymbol{U}_i^\top\right]_{(i,j)\in\Omega_{\boldsymbol{R}_{ij}}} - \lambda_{\boldsymbol{V}}\boldsymbol{V}_j = 0. \tag{2.48}$$

By solving the above two equations, we can get:

$$\boldsymbol{U}_i = [(\boldsymbol{V}_j\boldsymbol{V}_j^\top)_{j\in\Omega_{\boldsymbol{U}_i}} + \lambda_{\boldsymbol{U}}\boldsymbol{I}]^{-1}(\boldsymbol{R}_{ij}\boldsymbol{V}_j^\top)_{j\in\Omega_{\boldsymbol{U}_i}}, \tag{2.49}$$

$$\boldsymbol{V}_j = [(\boldsymbol{U}_i\boldsymbol{U}_i^\top)_{i\in\Omega_{\boldsymbol{V}_j}} + \lambda_{\boldsymbol{V}}\boldsymbol{I}]^{-1}(\boldsymbol{R}_{ij}\boldsymbol{U}_i^\top)_{i\in\Omega_{\boldsymbol{V}_j}}. \tag{2.50}$$

That is, the iterative solution of $\boldsymbol{U}$ and $\boldsymbol{V}$ are obtained. By calculating using the above iterative formulas, $\boldsymbol{U}$ and $\boldsymbol{V}$ obtained after convergence can be used to calculate the recommendation results.

(3) Bayesian Probabilistic Matrix Factorization

The probabilistic matrix factorization model involves fewer parameters and the estimation of the parameters is all point estimation, which is easy to cause the problem of overfitting in the process of model training. Therefore, Ruslan Salakhutdinov and Andriy Mnih improved the probabilistic matrix factorization by a full Bayesian inference to ensure that the model capacity is controlled by parameters and hyperparameters, that is, the Bayesian probabilistic matrix factorization [14].

In the Bayesian probabilistic matrix factorization, the posterior probability, likelihood function, and prior probability of the model parameters are basically consistent with the probabilistic matrix factorization, that is:

Posterior probability:

$$p(\theta \mid \boldsymbol{R}, \alpha) = \frac{p(\boldsymbol{R} \mid \theta, \alpha)p(\theta \mid \alpha)}{p(\boldsymbol{R} \mid \alpha)} \propto p(\boldsymbol{R} \mid \theta, \alpha)p(\theta \mid \alpha). \tag{2.51}$$

Likelihood function:

$$p(\boldsymbol{R} \mid \boldsymbol{U}, \boldsymbol{V}, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} [\mathcal{N}(\boldsymbol{R}_{ij} \mid \boldsymbol{U}_i^\top \boldsymbol{V}_j, \sigma^2)]^{I_{ij}}. \tag{2.52}$$

Prior probability:

$$p(\boldsymbol{U} \mid \mu_{\boldsymbol{U}}, \Lambda_{\boldsymbol{U}}) = \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{U}_i \mid \mu_{\boldsymbol{U}}, \Lambda_{\boldsymbol{U}}^{-1}), \tag{2.53}$$

$$p(\boldsymbol{V} \mid \mu_{\boldsymbol{V}}, \Lambda_{\boldsymbol{V}}) = \prod_{i=1}^{M} \mathcal{N}(\boldsymbol{V}_i \mid \mu_{\boldsymbol{V}}, \Lambda_{\boldsymbol{V}}^{-1}). \tag{2.54}$$

Based on the prior probability, the Bayesian probabilistic matrix factorization further introduces Gaussian-Wishart prior for the parameter $\Theta_{\boldsymbol{U}} = \{\mu_{\boldsymbol{U}}, \Lambda_{\boldsymbol{U}}\}$, $\Theta_{\boldsymbol{V}} = \{\mu_{\boldsymbol{V}}, \Lambda_{\boldsymbol{V}}\}$, namely:

$$\begin{aligned} p(\Theta_{\boldsymbol{U}} \mid \Theta_0) &= p(\mu_{\boldsymbol{U}} \mid \Lambda_{\boldsymbol{U}}) p(\Lambda_{\boldsymbol{U}}) = \mathcal{N}(\mu_{\boldsymbol{U}} \mid \mu_0, (\beta_0 \Lambda_{\boldsymbol{U}})^{-1}) \mathcal{W}(\Lambda_{\boldsymbol{U}} \mid \boldsymbol{W}_0, \nu_0), \\ p(\Theta_{\boldsymbol{V}} \mid \Theta_0) &= p(\mu_{\boldsymbol{V}} \mid \Lambda_{\boldsymbol{V}}) p(\Lambda_{\boldsymbol{V}}) = \mathcal{N}(\mu_{\boldsymbol{V}} \mid \mu_0, (\beta_0 \Lambda_{\boldsymbol{V}})^{-1}) \mathcal{W}(\Lambda_{\boldsymbol{V}} \mid \boldsymbol{W}_0, \nu_0), \end{aligned} \tag{2.55}$$

where $\mathcal{W}$ represents the Wishart distribution, the degree of freedom of this distribution is ν_0, and the range matrix is $\boldsymbol{W}_0 \in \mathbb{R}^{d \times d}$, additionally:

$$\mathcal{W}(\Lambda \mid \boldsymbol{W}_0, \nu_0) = \frac{1}{C} |\Lambda|^{(\nu_0 - D - 1)/2} \exp\left(-\frac{1}{2} \mathrm{Tr}(\boldsymbol{W}_0^{-1} \Lambda) \right), \tag{2.56}$$

where C represents the normalization constant.

After the above definition, the rating prediction based on the Bayesian probabilistic matrix factorization can be given by:

$$\begin{aligned} p(R_{ij}^* \mid \boldsymbol{R}, \Theta_0) = \iint & p(R_{ij}^* \mid \boldsymbol{U}_i, \boldsymbol{V}_j) p(\boldsymbol{U}, \boldsymbol{V} \mid \boldsymbol{R}, \Theta_{\boldsymbol{U}}, \Theta_{\boldsymbol{V}}) \\ & p(\Theta_{\boldsymbol{U}}, \Theta_{\boldsymbol{V}} \mid \Theta_0) \mathrm{d}\{\boldsymbol{U}, \boldsymbol{V}\} \mathrm{d}\{\Theta_{\boldsymbol{U}}, \Theta_{\boldsymbol{V}}\}. \end{aligned} \tag{2.57}$$

Due to the complexity of the posterior distribution, the above equation is unsolvable. Therefore, here we can use the Markov Chain Monte Carlo (MCMC) method to approximate the scoring prediction process, namely:

$$p(R_{ij}^* \mid \boldsymbol{R}, \Theta_0) \approx \frac{1}{K} \sum_{k=1}^{K} p(R_{ij}^* \mid \boldsymbol{U}_i^{(k)}, \boldsymbol{V}_j^{(k)}), \tag{2.58}$$

where $\boldsymbol{U}_i^{(k)}, \boldsymbol{V}_j^{(k)}$ represent the stationary distribution of $\{\boldsymbol{U}, \boldsymbol{V}, \Theta_{\boldsymbol{U}}, \Theta_{\boldsymbol{V}}\}$ based on model parameters and hyperparameters, respectively, generated by a Markov chain. Consider a simple MCMC algorithm, that is, the Gibbs sampling algorithm. Since the Bayesian probabilistic matrix factorization uses conjugate prior distributions for parameters and hyperparameters, the conditional distribution obtained from the posterior distribution is easy to sample. Based on the rating matrix $\boldsymbol{R}$, the item embedding vector $\boldsymbol{V}$, the parameter $\Theta_{\boldsymbol{U}}$, the hyperparameter α, the conditional distribution of the user embedding vector $\boldsymbol{U}_i$ is:

$$p(\boldsymbol{U}_i \mid \boldsymbol{R}, \boldsymbol{V}, \Theta_{\boldsymbol{U}}, \alpha) = \mathcal{N}(\boldsymbol{U}_i \mid \mu_i^*, [\Lambda_i^*]^{-1}) \sim \prod_{j=1}^{M} [\mathcal{N}(R_{ij} \mid \boldsymbol{U}_i^\top \boldsymbol{V}_j, \alpha^{-1})]^{I_{ij}} p(\boldsymbol{U}_i \mid \mu_{\boldsymbol{U}}, \Lambda_{\boldsymbol{U}}), \tag{2.59}$$

where

$$\Lambda_i^* = \Lambda_{\boldsymbol{U}} + \alpha \sum_{j=1}^{M} [\boldsymbol{V}_j \boldsymbol{V}_j^\top]^{I_{ij}}, \tag{2.60}$$

$$\mu_i^* = [\Lambda_i^*]^{-1} \left(\alpha \sum_{j=1}^{M} [\boldsymbol{V}_j R_{ij}]^{I_{ij}} + \Lambda_{\boldsymbol{U}} \mu_{\boldsymbol{U}} \right). \tag{2.61}$$

In addition, the hyperparameters $\mu_{\boldsymbol{U}}, \Lambda_{\boldsymbol{U}}$ are generated by sampling from a Wishart-Gaussian distribution:

$$p(\mu_{\boldsymbol{U}}, \Lambda_{\boldsymbol{U}} \mid \boldsymbol{U}, \Theta_0) = \mathcal{N}(\mu_{\boldsymbol{U}} \mid \mu_0^*, (\beta_0^* \Lambda_{\boldsymbol{U}})^{-1}) \mathcal{W}(\Lambda_{\boldsymbol{U}} \mid \boldsymbol{W}_0^*, \nu_0^*), \tag{2.62}$$

where

$$\mu_0^* = \frac{\beta_0 \mu_0 + N\bar{U}}{\beta_0 + N}, \beta_0^* = \beta_0 + N, \nu_0^* = \nu_0 + N, \tag{2.63}$$

$$[\boldsymbol{W}_0^*]^{-1} = \boldsymbol{W}_0^{-1} + N\bar{S} + \frac{\beta_0 N}{\beta_0 + N} (\mu_0 - \bar{U})(\mu_0 - \bar{U})^\top, \tag{2.64}$$

$$\bar{U} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{U}_i, \bar{S} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{U}_i \boldsymbol{U}_i^{\top}. \tag{2.65}$$

Based on the above formulas, we can obtain the update process of the user embedding vector. And the update process of the item embedding vector is similar to that of the users, so it will not be repeated here.

Factorization Machine

Factorization Machine (FM) [13] is a method for modeling the relationship between users and items through polynomials, proposed by Steffen Rendle in 2010. FM's polynomial model incorporates the idea of matrix factorization, that is, the coefficients of the second-order cross feature terms are adjusted in a matrix factorization manner, so that the coefficients are no longer independent and irrelevant. And at the same time, it can solve the problem that the parameters are not sufficiently trained due to data sparsity. The second-order FM model is expressed as follows:

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j, \tag{2.66}$$

where n represents the number of features of the data sample; x_i represents the value of the *i*-th feature; w_0, w_i, w_{ij} represent the parameters of the model. In practical recommender systems, there are usually serious data sparsity problems, which bring great challenges to the training of FM models. Sparse data will lead to insufficient training samples for cross feature terms, so the parameters w_{ij} obtained by training do not meet the characteristics of sufficient statistics, resulting in inaccurate parameters w_{ij}, which in turn affects the performance of model prediction. In order to solve the above training problem, FM draws on the idea of matrix factorization to form a symmetric matrix $\boldsymbol{W}$ of all quadratic parameters w_{ij}. Then this matrix can be decomposed into $\boldsymbol{W} = \boldsymbol{V}^{\top}\boldsymbol{V}$, where the *j*-th column of $\boldsymbol{V}$ is the hidden vector of the *j*-th feature v_j. Each parameter $w_{ij} =< \boldsymbol{v}_i, \boldsymbol{v}_j >$, so the original FM expression can be written as:

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \boldsymbol{v}_i, \boldsymbol{v}_j > x_i x_j, \tag{2.67}$$

where $\boldsymbol{v}_i$ is the hidden vector of the *i*-th feature; $< \cdot, \cdot >$ represents the vector dot product. The length of the hidden vector is $k(k \ll n)$, which means that k dimensional vectors are used to describe user features. The first term of the second-order FM expression represents the global bias, the second term represents the linear relationship between input and output, and the third term is the second-order cross term, indicating that the model considers the interaction between two different features, thereby establishing a bilinear relationship between input and output. If the cross-term coefficient is 0, it means that the corresponding two features have no correlation. Such a design can reduce the redundancy of the model and improve the predictive ability of the model.

The Connections and Differences Between Matrix Factorization and Factorization Machines

Factorization machines can be thought of as an extension of matrix factorization. A factorization machine is equivalent to a matrix factorization model if only quadratic terms are kept in the factorization machine model. And, the model optimization of the factorization machine also follows the optimization method of probabilistic matrix factorization or Bayesian probabilistic matrix factorization. For instance, the stochastic gradient descent method used in the factorization machine learns the model in the same way as the classical matrix factorization method, and the factorization machine can use the Markov chain Monte Carlo method to learn the model in the same way as the Bayesian probabilistic matrix factorization method.

There are two main differences between the two algorithms. One is that the factorization machine can use more information. It cannot only use the rating information of users on items but also use a lot of additional information, such as user attributes, item characteristics, social network, and context information. With this information, factorization machine can provide effective recommendations even in the face of new users or new items. However, most matrix factorization methods are difficult to directly utilize these additional information and cannot solve the cold-start problem. The second is that factorization machine can model more complex feature

relationships. In addition to the same quadratic terms as matrix factorization, factorization machine contains constant and linear terms and can even contain higher-order feature interaction terms such as cubic and quartic terms. Compared with the matrix factorization models, the factorization machine can significantly improve the capacity and expressive ability of the model, which helps to improve the accuracy of prediction.

2.3 Summary

This chapter introduces four types of classic recommendation algorithms, including content-based recommendation algorithms, classic collaborative filtering algorithms, matrix factorization methods, and factorization machines. Before the emergence of deep learning, these methods were the most mainstream techniques for recommender systems, widely recognized by both academia and industry. Although after the emergence of deep learning, these technologies are no longer the first choice of the industry, but the basic ideas and practical experience extracted from these technologies still affect the follow-up research. Therefore, in many deep learning-based recommendation algorithms, we can often see the reflections of the above approaches.

References

1. Shumeet Baluja et al. “Video Suggestion and Discovery for YouTube: Taking Random Walks through the View Graph”. In: *Proceedings of the 17th International Conference on World Wide Web*. WWW ’08. Beijing, China: Association for Computing Machinery, 2008, pp. 895–904. ISBN: 9781605580852. URL: https://doi.org/10.1145/1367497.1367618.
2. Tom Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 33*. Ed. by H. Larochelle et al. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64aPaper.pdf.
3. Yao Cheng, Liang Yin, and Yong Yu. “LorSLIM: Low Rank Sparse Linear Methods for Top-N Recommendations”. In: *2014 IEEE International Conference on Data Mining*. 2014, pp. 90–99. DOI: https://doi.org/10.1109/ICDM.2014.112.
4. Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019,

pp. 4171–4186. DOI: https://doi.org/10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

5. Li Dong et al. “Unified Language Model Pre-training for Natural Language Understanding and Generation”. In: *Advances in Neural Information Processing Systems 32. 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 13019–13031. URL: https://proceedings.neurips.cc/paper/2019/file/c20bb2d9a50d5ac1f713f8b34d9aac5aPaper.pdf.

6. Jonathan L. Herlocker et al. “An Algorithmic Framework for Performing Collaborative Filtering”. In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’99. Berkeley, California, USA: Association for Computing Machinery, 1999, pp. 230–237. ISBN: 1581130961. URL: https://doi.org/10.1145/312624.312682.

7. Quoc Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Beijing, China: PMLR, June 2014, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html.

8. Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

9. Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: https://doi.org/10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

10. Xia Ning and George Karypis. “SLIM: Sparse Linear Methods for Top-N Recommender Systems”. In: *2011 IEEE 11th International Conference on Data Mining*. 2011, pp. 497–506. DOI: https://doi.org/10.1109/ICDM.2011. 134.

11. Xia Ning and George Karypis. “Sparse Linear Methods with Side Information for Top-n Recommendations”. In: *Proceedings of the Sixth ACM Conference on Recommender Systems*. RecSys ’12. Dublin, Ireland: Association for Computing Machinery, 2012, pp. 155–162. ISBN: 9781450312707. DOI: https://doi.org/10.1145/2365952.2365983.

12. Michael J. Pazzani and Daniel Billsus. “Content-Based Recommendation Systems”. In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. ISBN: 978-3-540-72079-9. DOI: https://doi.org/10.1007/978-3-540-72079-9_10.

13. Steffen Rendle. “Factorization Machines”. In: *2010 IEEE International Conference on Data Mining*. 2010, pp. 995–1000. DOI: https://doi.org/10.1109/ICDM.2010.127.

14. Ruslan Salakhutdinov and Andriy Mnih. “Bayesian Probabilistic Matrix Factorization Using Markov Chain Monte Carlo”. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML ’08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 880–887. ISBN: 9781605582054. DOI: https://doi.org/10.1145/1390156.1390267.

15. Russ R Salakhutdinov and Andriy Mnih. “Probabilistic Matrix Factorization”. In: *Advances in Neural Information Processing Systems 20. 21st Annual Conference on Neural Information Processing Systems 2007*. Ed. by J. Platt et al. Curran Associates, Inc., 2007, pp. 1257–1264.

URL: https://proceedings.neurips.cc/paper/2007/file/d7322ed717dedf1eb4e6e52a37ea7bcd-Paper.pdf.

16. Badrul Sarwar et al. “Item-Based Collaborative Filtering Recommendation Algorithms”. In: *Proceedings of the 10th International Conference on World Wide Web*. WWW ’01. Hong Kong, Hong Kong: Association for Computing Machinery, 2001, pp. 285–295. ISBN: 1581133480. URL: https://doi.org/10.1145/371920.372071.

17. Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30. 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, California, USA: Curran Associates, Inc., 2017, pp. 6000–6010. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_3

3. Foundations of Deep Learning

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter introduces the basics of deep learning, including feedforward computation and backpropagation algorithms for deep neural networks, as well as various classic neural network models. As readers learn, they can combine the content of other chapters in this book to understand and design different types of neural network models for recommendation scenarios, taking into account the data characteristics and task properties, in order to improve recommendation performance.

Keywords Deep learning basics – Feedforward computation – Backpropagation – Deep neural networks

This chapter first introduces the feedforward computation and back-propagation algorithms of neural networks, helping readers better grasp the knowledge of deep learning, understand its optimization operations, and assist in designing recommendation models suitable for various recommendation scenarios. Then, this chapter introduces various deep learning models, including multi-layer neural networks, convolutional neural networks, recurrent neural networks, attention mechanisms,

sequence modeling, and pre-training. These models play a very important role in various tasks in the recommendation scenario.

3.1 Neural Networks and Feedforward Computation

This section first introduces the basic structure of a Neural Network (NN), and then it gives an example to explain the feedforward computation process of a neural network. A neural network, also known as an Artificial Neural Network (ANN), is a computational model that mimics the structure of biological neural networks. In a neural network, each neuron acts as a basic computation unit, receiving a certain number of input signals from other neurons, processing them, and then transmitting them to other neurons.

Typically, the signals transmitted between neurons are real numbers. Each neuron first calculates the weighted sum of the input signals from the previous layer, where the weights are represented by $w_0, \cdots, w_{n-1}$, and adds a bias term *b*. It then passes the result through a non-linear activation function $f(\cdot)$ before transmitting it to the next layer of neurons. This process can be represented by the diagram in Fig. 3.1.

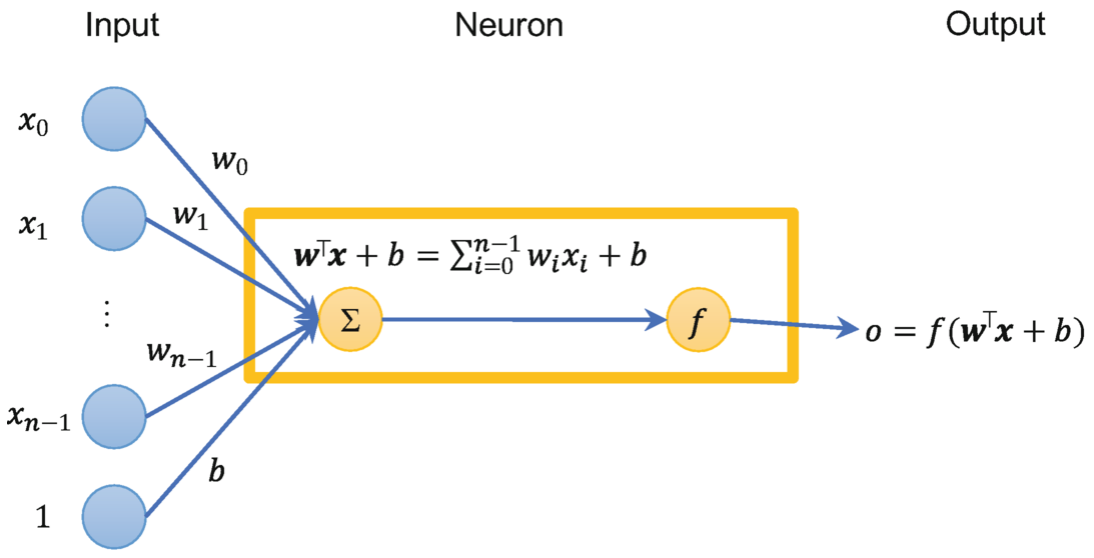


Fig. 3.1 An example of neuron structure in a neural network

The feedforward neural network (FNN) refers to a neural network where the neurons are connected to form a neural network without loops. The most common feedforward neural network model is the perceptron model, which was first proposed by Frank Rosenblatt in 1958. A single-layer perceptron is the simplest feedforward neural network model, and its structure is the same as in Fig. 3.1. A common example of a single-layer perceptron model is a single-layer neural network combined with a Logistic function (also known as the Sigmoid function), that is,

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}}.$$

When this function is used as a neural network activation function, the single-layer perceptron becomes a Logistic model. Common activation functions include ReLU, tanh, and others.

This single-layer perceptron model is often used as the basic building block of more complex neural networks, and it can be used to classify linear separable data.

The feedforward calculation of a neural network refers to the process of computing the corresponding output for a given input of the neural network. For example, in the Logistic model, given the input $\boldsymbol{x}$, the process of computing the corresponding output

$$y = \frac{1}{1 + \mathrm{e}^{-(\boldsymbol{w}^\top \boldsymbol{x} + b)}} \tag{3.1}$$

is called the feedforward calculation of the neural network.

The following introduces the model structure, parameters, and forward calculation process of a two-layer perceptron. As shown in Fig. 3.2, the model contains a hidden layer composed of neurons. The output of the model can be represented as

$$\boldsymbol{h} = f_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) \tag{3.2}$$

$$\begin{aligned} \boldsymbol{o} &= f_2(\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2) \\ &= f_2(\boldsymbol{W}_2 f_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2). \end{aligned} \tag{3.3}$$

In the above equations, $\boldsymbol{W}_1 \in \mathbb{R}^{k \times n}$ and $\boldsymbol{b}_1 \in \mathbb{R}^k$ are the weight matrix and bias term from the input layer to the hidden layer, respectively; $\boldsymbol{W}_2 \in \mathbb{R}^{m \times k}$ and $\boldsymbol{b}_2 \in \mathbb{R}^m$ are the weight matrix and bias term from the

hidden layer to the output layer, respectively; f_1 and f_2 are the activation functions of the two layers. The forward calculation process of the model is the process of calculating the hidden layer output $\boldsymbol{h}$ and the model output $\boldsymbol{o}$ given input $\boldsymbol{x}$.

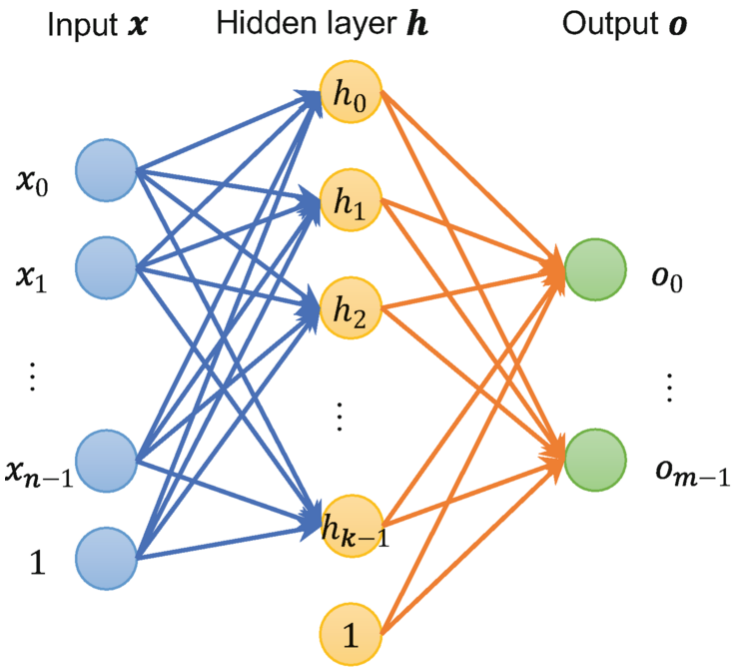


Fig. 3.2 The structure of the two-layer network model

3.2 Back-Propagation Algorithm

The next chapter will introduce how to optimize neural network models. Here, using the model in Fig. 3.2 as an example, the Back-Propagation (BP) algorithm in neural network optimization will be introduced [3]. Back-propagation algorithm is a widely used algorithm in the training of feedforward neural networks. In the process of fitting neural networks, the back-propagation algorithm can efficiently calculate the gradient of the loss function with respect to the parameters of the neural network and use the gradient descent method to update the parameters such that the loss function of the neural network is minimized.

Back-propagation refers to the calculation of gradients of each layer in a reverse direction starting from the loss function, for the purpose of gradient updates and optimization. In the example of a two-layer neural network

above, the gradients of the loss function with respect to the parameters $\boldsymbol{W}_2$ and $\boldsymbol{b}_2$ are first calculated. As shown in the model in Fig. 3.2, assuming that the input $\boldsymbol{x}$ corresponds to the true label $\boldsymbol{y}$, the output is $\boldsymbol{o}$, and the loss function is $L(\boldsymbol{y}, \boldsymbol{o})$, where $\boldsymbol{z}_2 = \boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2$, the gradient of the loss function with respect to the weight matrix $\boldsymbol{W}_2$ for the (i, j) element $\boldsymbol{W}_{2,ij}$ can be calculated through the chain rule:

$$\frac{\partial L}{\partial \boldsymbol{W}_{2,ij}} = \frac{\partial L}{\partial \boldsymbol{o}_i} \cdot \frac{\partial \boldsymbol{o}_i}{\partial \boldsymbol{W}_{2,ij}} = \frac{\partial L}{\partial \boldsymbol{o}_i} \cdot \frac{\partial f_2(\boldsymbol{z}_{2,i})}{\partial \boldsymbol{z}_{2,i}} \cdot \frac{\partial \boldsymbol{z}_{2,i}}{\partial \boldsymbol{W}_{2,ij}} = \frac{\partial L}{\partial \boldsymbol{o}_i} \cdot f_2'(\boldsymbol{z}_{2,i}) \cdot \boldsymbol{h}_j. \tag{3.4}$$

In the equation, $\frac{\partial L}{\partial \boldsymbol{o}_i}$ is the gradient of the loss function with respect to the output $\boldsymbol{o}_i$, $f_2'(\boldsymbol{z}_{2,i})$ is the gradient with the activation function f_2, and $\boldsymbol{h}_j$ is the output of the hidden layer; the product of these three terms can derive the gradient of the loss function with respect to $\boldsymbol{W}_{2,ij}$. This formula can be written in the form of matrix multiplication

$$\frac{\partial L}{\partial \boldsymbol{W}_2} = \left(\frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2' \right) \cdot \boldsymbol{h}^\top. \tag{3.5}$$

In the equation,
$\boldsymbol{h} \in \mathbb{R}^k$, $\frac{\partial L}{\partial \boldsymbol{o}} \in \mathbb{R}^m$, $\boldsymbol{f}_2' = [f_2'(\boldsymbol{z}_{2,0}), \cdots, f_2'(\boldsymbol{z}_{2,m-1})]^\top \in \mathbb{R}^m$; $\odot$ represents the element-wise multiplication. The final product result $\frac{\partial L}{\partial \boldsymbol{W}_2} \in \mathbb{R}^{m \times k}$ is the gradient of the loss function with respect to the entire weight matrix $\boldsymbol{W}_2$.

Similarly, the gradient of the loss function with respect to the bias term $\boldsymbol{b}_2$ can also be calculated using the chain rule.

$$\frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2'. \tag{3.6}$$

In the computation process, we can first calculate the gradients of $\frac{\partial L}{\partial \boldsymbol{o}}$ and $\boldsymbol{f}_2'$, then calculate the gradient of $\frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2'$, and finally use the hidden layer output $\boldsymbol{h}$ and the already calculated gradient $\frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2'$ to directly obtain the gradient of the term $\frac{\partial L}{\partial \boldsymbol{W}_2}$.

Next, we will calculate the gradient of the loss function with respect to the parameters $\boldsymbol{W}_1$ and $\boldsymbol{b}_1$ of the first layer in the neural network. This process is a little bit more complex.

$$
\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{W}_{1,ij}} &= \left(\left(\frac{\partial L}{\partial \boldsymbol{h}} \right)^{\top} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{W}_{1,ij}} \right) \\
&= \left(\left(\frac{\partial L}{\partial \boldsymbol{o}} \right)^{\top} \cdot \frac{\partial \boldsymbol{o}}{\partial (\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2)} \cdot \frac{\partial (\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2)}{\partial \boldsymbol{h}} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{W}_{1,ij}} \right) \\
&= \left(\left(\frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2' \right)^{\top} \cdot \boldsymbol{W}_2 \cdot \frac{\partial \boldsymbol{h}}{\partial (\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)} \cdot \frac{\partial (\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)}{\partial \boldsymbol{W}_{1,ij}} \right) \\
&= \left(\left[\left(\left(\frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2' \right)^{\top} \cdot \boldsymbol{W}_2 \right) \odot \boldsymbol{f}_1' \right]_{(i)} \cdot \boldsymbol{x}_j \right).
\end{aligned}
\tag{3.7}
$$

In this equation, $\left[\cdots \right]_{(i)}$ represents the *i*-th element in the vector. The above equation can also be written in the matrix form as

$$
\frac{\partial L}{\partial \boldsymbol{W}_1} = \left(\left(\frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2' \right)^{\top} \cdot \boldsymbol{W}_2 \right) \odot \boldsymbol{f}_1' \cdot \boldsymbol{x}^{\top}. \tag{3.8}
$$

It can be seen that the gradient of the loss function with respect to $\boldsymbol{W}_1$ contains the $\frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}_2'$ term that has been previously calculated. The method for calculating the gradient of the loss function with respect to $\boldsymbol{b}_1$ is also similar and will not be discussed further here.

Up to this point, this section has explained how to calculate the gradient of the loss function with respect to each parameter, layer by layer, from the final output layer backward. Taking the mean square error (MSE) function as an example, the specific process for calculating the gradient of the loss function with respect to the parameters of a *K*-layer neural network is as follows:

Step 1: First, perform feedforward computation on the input of the neural network to obtain the output of each layer:

$$
\boldsymbol{z}_1 = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1, \boldsymbol{a}_1 = f_1(\boldsymbol{z}_1), \boldsymbol{z}_2 = \boldsymbol{W}_2 \boldsymbol{a}_1 + \boldsymbol{b}_2, \cdots, \boldsymbol{o} = f_K(\boldsymbol{z}_K). \tag{3.9}
$$

Step 2: Compute the gradient of the loss function with respect to the output:

$$L(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{o})^2, \frac{\partial L}{\partial \boldsymbol{o}} = \boldsymbol{y} - \boldsymbol{o}. \tag{3.10}$$

Step 3: Compute the gradient of the activation function of the last layer of the neural network:

$$\frac{\partial f_K(\boldsymbol{z}_K)}{\partial \boldsymbol{z}_K} = f'_K(\boldsymbol{z}_K) = f'_K. \tag{3.11}$$

Let

$$\boldsymbol{\delta}_K = \frac{\partial L}{\partial \boldsymbol{o}} \odot \boldsymbol{f}'_K. \tag{3.12}$$

In this equation, $\boldsymbol{\delta} K$ represents the error term for this layer. Next, compute the gradient of the loss function with respect to the parameters $\boldsymbol{W}_K$ and $\boldsymbol{b}_K$:

$$\frac{\partial L}{\partial \boldsymbol{W}_K} = \boldsymbol{\delta}_K \boldsymbol{z}_K^\top, \frac{\partial L}{\partial \boldsymbol{b}_2} = \boldsymbol{\delta}_K. \tag{3.13}$$

Step 4: Compute the gradient of the activation function of the second-to-last layer of the neural network:

$$\frac{\partial f_{K-1}(\boldsymbol{z}_{K-1})}{\partial \boldsymbol{z}_{K-1}} = f'_{K-1}(\boldsymbol{z}_{K-1}) = f'_{K-1}. \tag{3.14}$$

Let

$$\boldsymbol{\delta}_{K-1} = (\boldsymbol{\delta}_{K-1}^\top \boldsymbol{W}_K) \odot \boldsymbol{f}'_{K-1}. \tag{3.15}$$

Compute the gradient of the loss function with respect to the parameters $\boldsymbol{W}_{K-1}, \boldsymbol{b}_{K-1}$:

$$\frac{\partial L}{\partial \boldsymbol{W}_{K-1}} = \boldsymbol{\delta}_{K-1} \boldsymbol{z}_{K-1}^\top, \frac{\partial L}{\partial \boldsymbol{b}_{K-1}} = \boldsymbol{\delta}_{K-1}. \tag{3.16}$$

For a neural network with many layers, people can repeat steps 3 and 4 until the error term propagates from the output layer to the first layer, so as to obtain the gradient of the loss function with respect to all parameters. The algorithm of propagating the error term from the output layer to the input layer is the back-propagation algorithm. Next, optimization methods such

as gradient descent can be used to update the parameters, in order to achieve the goal of training the neural network.

3.3 Various Types of Deep Neural Networks

In this chapter, various types of deep neural networks will be introduced. In many fields such as information retrieval and recommendation, these different types of neural networks are used to help the systems better use heterogeneous information sources, including images of the items to be recommended, text of item reviews, users' sequential browsing behavior, etc. These models will be used in the other chapters of this book and will be explained and discussed here in a unified manner.

3.3.1 Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of deep neural network that is suitable for analyzing grid-like topology data, such as image data. The basic structure of CNN includes convolutional layers, activation layers, pooling layers, and fully connected layers, as shown in Fig. 3.3. This section will introduce the components of CNN in turn and give examples of the application of CNN in recommendation systems.

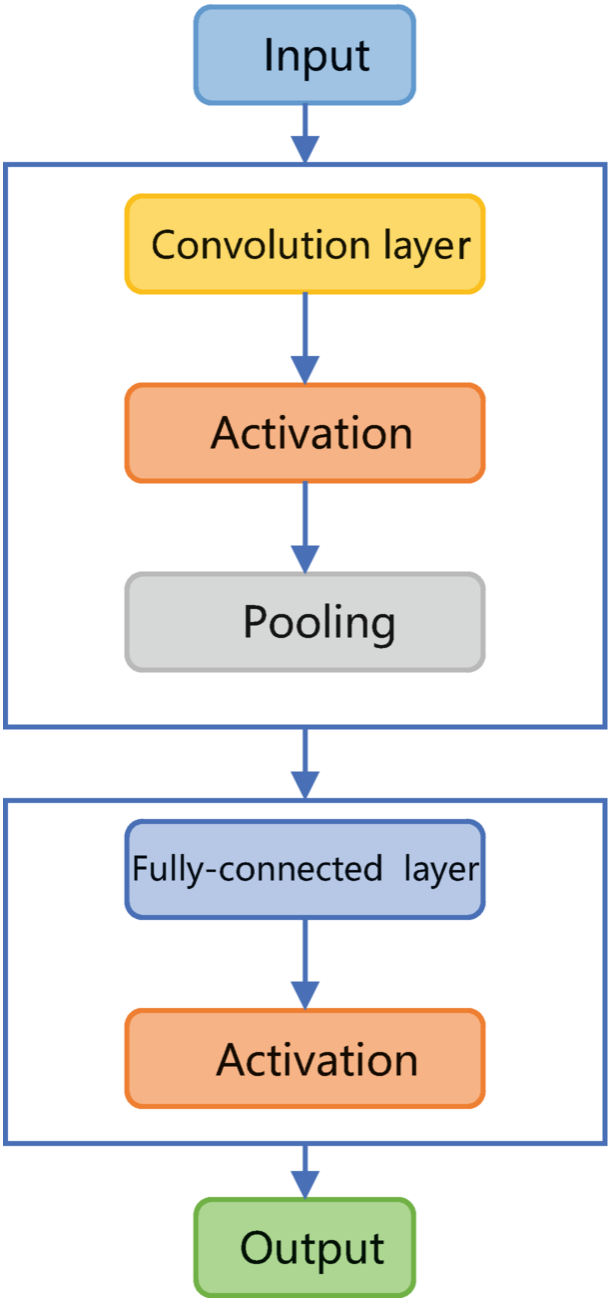


Fig. 3.3 The basic components of CNN: the combination of the basic components will be utilized multiple times

In conventional feedforward neural networks, a two-dimensional image (a matrix of pixel values) input is folded (i.e., flattened and concatenated) into a one-dimensional vector, which may cause the image to lose the information of its spatial structure. Moreover, because each pixel in the image is connected to a neuron in the network, the number of computation parameters is large. Compared to the traditional feedforward neural networks, CNNs use convolution operations at least in one layer to replace

general matrix multiplications, which helps to capture the dependency of spatial regions in the data, and to share computation parameters. This convolution operation layer is called convolutional layer. Because the design of CNNs is inspired by biological processes, the organization of neural connections in the visual cortex of animals is similar to artificial neural networks, and a single cortical neuron responds only to the limited regions of stimuli in the visual field, which is called the receptive field (RF). Different neurons' RFs partially overlap, making them cover the entire input image or the visual field of the input matrix.

The mathematical definition of convolution is a linear operation that takes two functions as inputs and produces a single function output. The one-dimensional convolution of two functions, $x(t)$ and $w(t)$, is defined as

$$s(t) = \int x(a)w(t-a)\mathrm{d}a \tag{3.17}$$

in continuous space and

$$s(t) = \sum_{a=-\infty}^{+\infty} x(a)w(t-a) \tag{3.18}$$

in discrete space.

Take the example of a falling ball to illustrate the meaning of the formula (Fig. 3.4). Assume that a ball falls from the air and it will undergo one-dimensional motion. The probability that the ball will land at a distance of *a* units from the starting point after the first fall is $x(a)$, where *x* is the probability distribution function. After the first fall, pick up the ball and drop it from another height above where it first landed. The probability that the ball will reach a point *b* units away from the new starting point is $w(b)$, where *w* is a different probability distribution. As shown in the figure, if it is known that the position reached after the first fall is *a*, then the probability value of reaching *t* is $w(t-a)$. To consider all the possibilities of the ball reaching *t*, all possible combinations of reaching *t* are divided into two moves and the probability of each way is summed, which is $s(t)$.

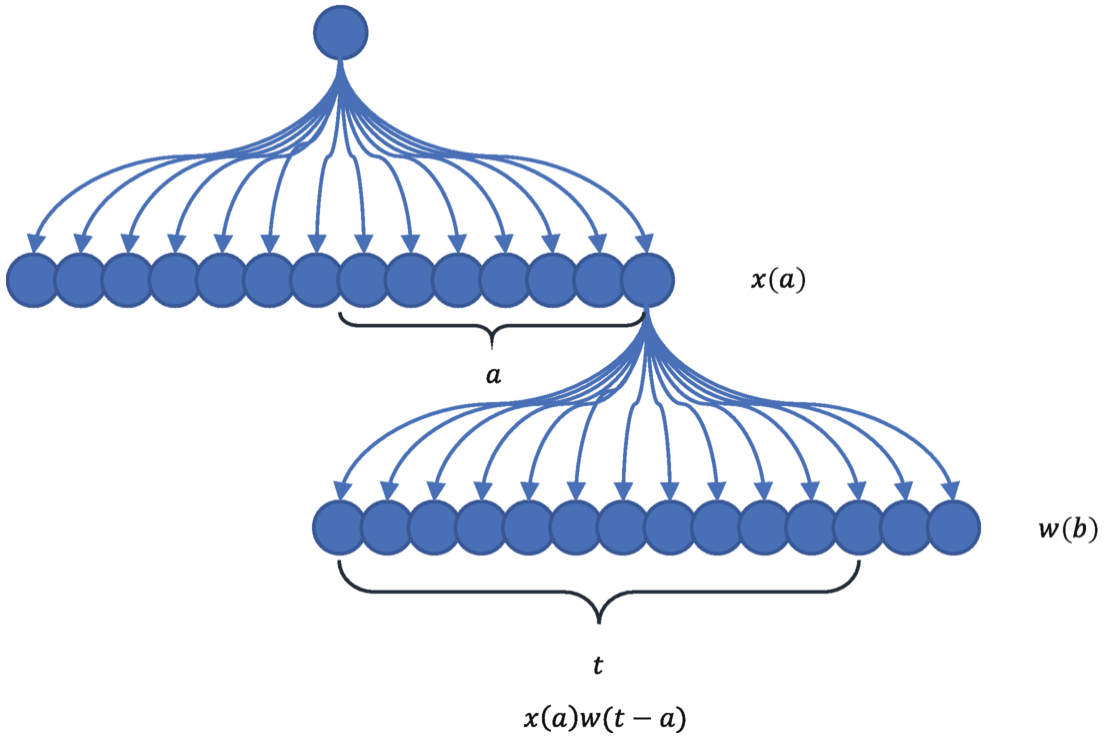


Fig. 3.4 Visualization of the probability of the ball falling into point *t*

As shown in Fig. 3.5, under discrete conditions, the probability of the ball falling a distance of *t*, i.e., $s(t)$, is calculated by shifting $w(b)$, multiplying it by the corresponding position of $x(a)$, and adding them up.

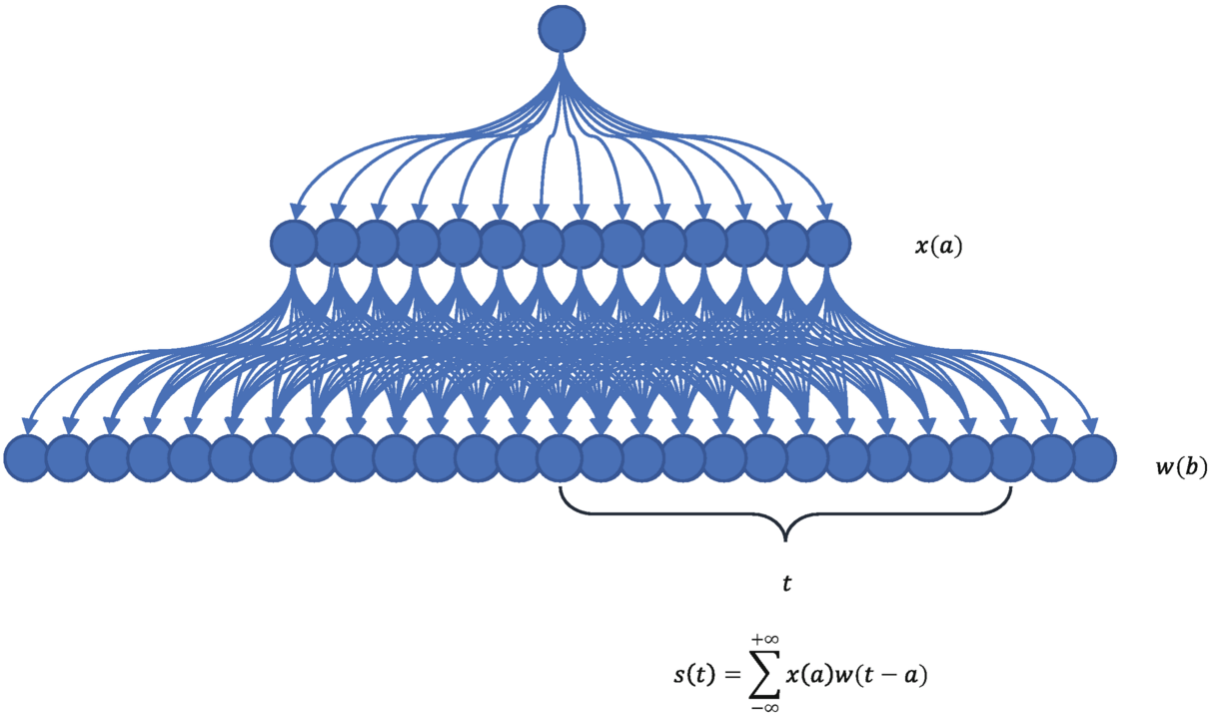


Fig. 3.5 Probability calculation of the ball falling into point *t*

In convolution, $x(t)$ can be considered as the input and $w(t)$ is the kernel function (weighting function) that weights $x(t)$. Similarly, discrete two-dimensional convolution can be extended from one-dimensional convolution:

$$s(i,j) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x(m,n)w(i-m, j-n). \tag{3.19}$$

Like one-dimensional convolution, two-dimensional convolution can be thought of as sliding over another function, performing multiplication and addition operations. This is the most common application of convolution in convolutional neural networks, where two-dimensional matrices (such as images) are viewed as two-dimensional functions and then a local function, called a “convolution kernel”, is used to convolute the image function. However, what is actually used in convolutional networks is not the original definition of convolution, but the cross-correlation function, which is a “convolution” without flipping operation:

$$s(i,j) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x(m,n)w(i+m, j+n). \tag{3.20}$$

Figure 3.6 shows an example of convolution operation on a two-dimensional matrix input. The convolution kernel slides over each element of the matrix and calculates the weighted sum of the neighboring elements with the convolution kernel as a new element value. This is the most basic convolution operation in a convolutional neural network, but in practice additional operations are often applied. Convolution kernels with size greater than 1 will result in a dimensionality reduction of the generated feature map compared to the input matrix. To retain the dimensionality of the input matrix and preserve more information at the edges, padding is often applied to the edges of the input matrix. Stride refers to the number of elements the convolution kernel skips when sliding, when the stride is equal to 1, it is a normal convolution operation, and when the stride is equal to 2, it means the convolution kernel will skip two elements before performing the convolution operation. Dilated convolution increases the receptive field in convolution by increasing the spacing between values processed by the convolution kernel.

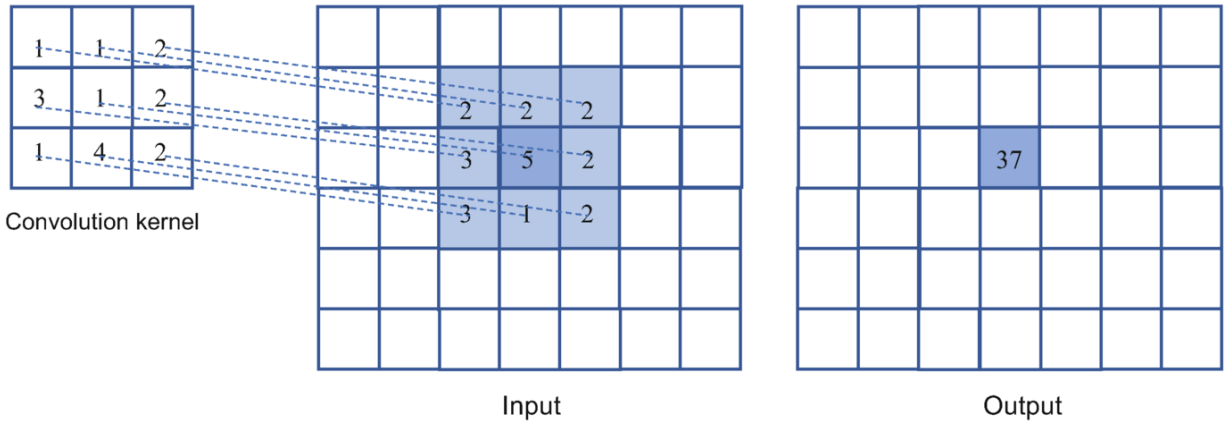


Fig. 3.6 Visualization of the calculation of convolution on a two-dimensional matrix

Figure 3.7 shows a 3×3 convolution with a dilation rate of 2. It has the same receptive field as a traditional 5×5 convolution, but the dilated convolution kernel only has 9 parameters, less than the 25 parameters of the 5×5 convolution kernel. This means that a wider field of view can be perceived at the same computational cost, thus improving the performance of the model.

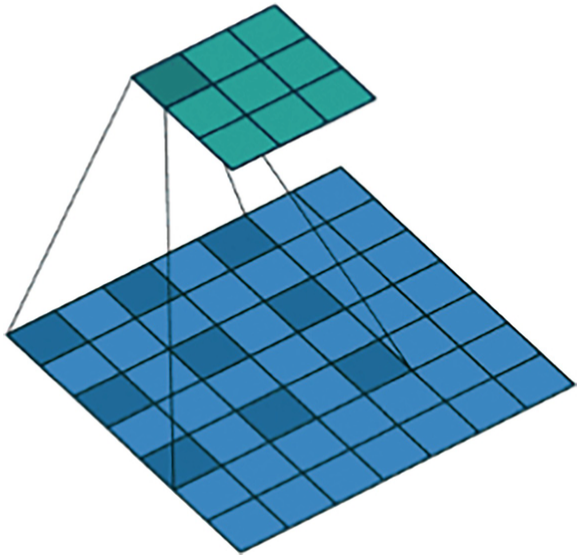


Fig. 3.7 Dilated convolution with dilation rate as 2

Since the output of a convolution layer is sensitive to the position of the features in the input, pooling layers are used in convolutional neural networks to aggregate the feature values in a region of a feature map with down-sampling, thus achieving translation invariance. Figure 3.8 shows two common pooling methods, average pooling and max pooling, which calculate the average and maximum values of all the elements in a pooling

window, respectively. Therefore, pooling layers are generally used in combination with convolutional layers.

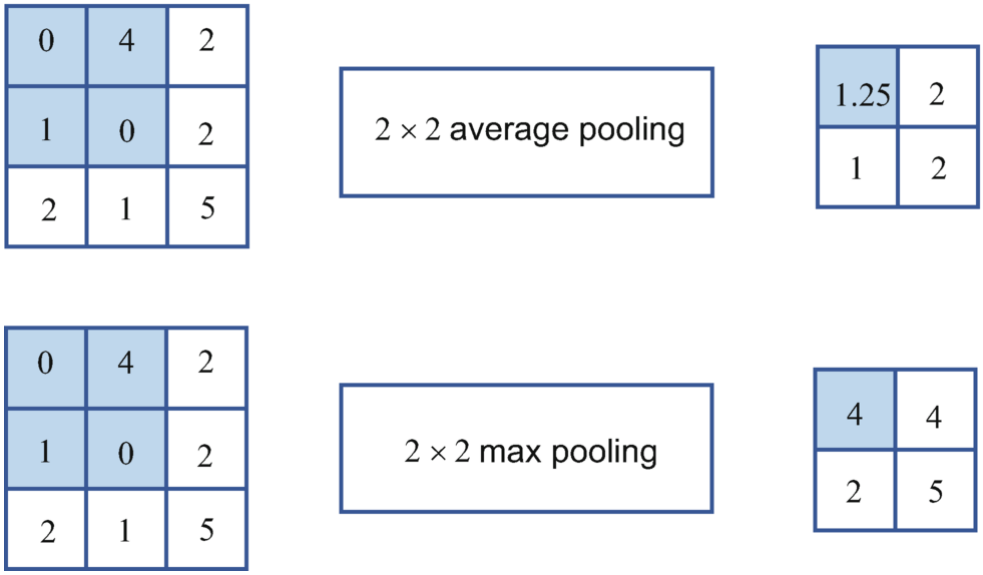


Fig. 3.8 Average pooling and max pooling

The activation functions and fully connected layers in convolutional neural networks have few differences from those in feedforward neural networks.

As convolutional neural networks are widely used in the image analysis, people have also studied their applications in natural language processing and recommendation systems. As shown in Fig. 3.9, by concatenating user behavior feature vectors at a single time point (such as embedding vectors of the watched movies or the purchased items) in temporal order from left to right into a 2D matrix, short-term behavior features of the user can be extracted by sliding convolution on the time dimension of the matrix, which can be used for modeling user sequential behavior. For more detailed recommendation algorithms, please refer to Sect. 4.4 of this book on sequential recommendation systems.

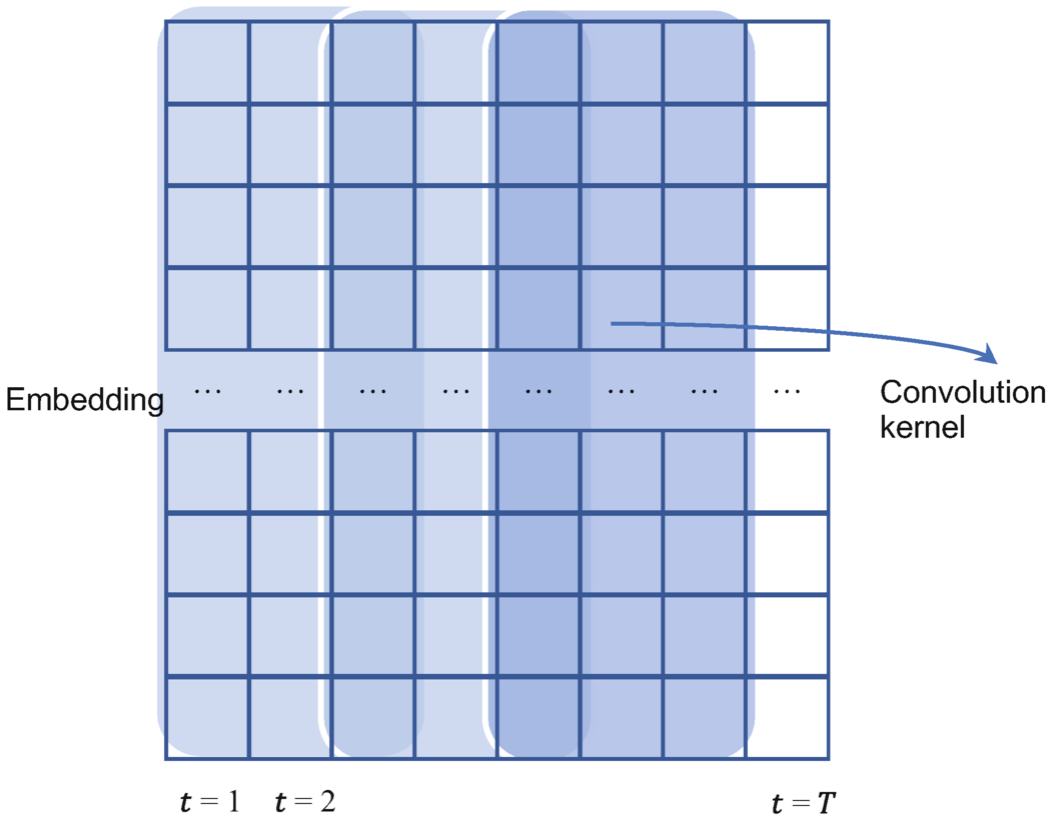


Fig. 3.9 Using CNN for feature extraction from temporal user behavior sequences

3.3.2 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a type of neural network that is used to process sequence-type data. This section will introduce three types of RNNs: traditional recurrent neural network, Long-Short Term Memory (LSTM), and Gate Recurrent Unit (GRU).

One of the key characteristics of RNNs is that they accept a sequence of inputs $\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^T$ and process each data element $\boldsymbol{x}^t$ in the sequence in order. When processing $\boldsymbol{x}^t$, the RNN takes into account the previous hidden state $\boldsymbol{h}^{t-1}$ in order to calculate the current hidden state $\boldsymbol{h}^t$ and output $\boldsymbol{o}^t$, as shown in the diagram (Fig. 3.10).

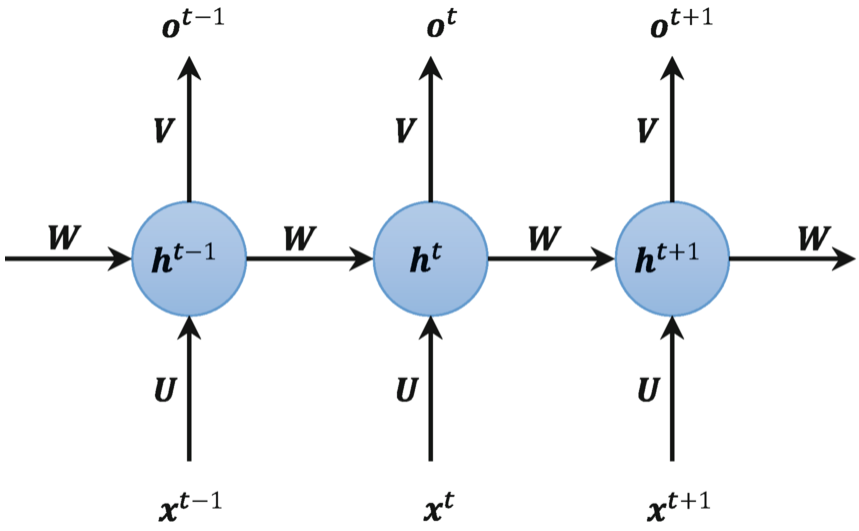


Fig. 3.10 Traditional RNN

The computation process of traditional recurrent neural networks is as follows:

$$\boldsymbol{a}^t = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{t-1} + \boldsymbol{U}\boldsymbol{x}^t \tag{3.21}$$

$$\boldsymbol{h}^t = \tanh(\boldsymbol{a}^t) \tag{3.22}$$

$$\boldsymbol{o}^t = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^t. \tag{3.23}$$

In the equation, $\boldsymbol{b}$ and $\boldsymbol{c}$, respectively, represent the bias vectors of the corresponding neural network layer. It should be noted that for the input at different times in the sequence, the recurrent neural network uses the same parameters for calculation, that is, the input $\boldsymbol{x}^t$ at different time steps requires the same parameters $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ to be shared. Since the same parameters are used at each step, according to the definition of the back-propagation chain rule discussed earlier in this chapter, the gradients of the recurrent neural network tend to disappear or explode during the backward propagation along the time dimension. Specifically, assuming that there is a loss function L at time step t, the gradient of $\boldsymbol{h}^t$ is

$$\frac{\partial L}{\partial \boldsymbol{h}^t} = \boldsymbol{V}^\top \frac{\partial L}{\partial \boldsymbol{o}^t}. \tag{3.24}$$

Furthermore, considering the gradient backpropagated to time step $t-1$, the gradient is

$$\frac{\partial L}{\partial \boldsymbol{h}^{t-1}} = \left(\frac{\partial \boldsymbol{h}^{t}}{\partial \boldsymbol{h}^{t-1}}\right)^{\top} \frac{\partial L}{\partial \boldsymbol{h}^{t}} = \boldsymbol{W}^{\top} \frac{\partial L}{\partial \boldsymbol{h}^{t}} \tanh'(\boldsymbol{a}^{t}). \tag{3.25}$$

It is easy to find from the above equation that as we continue to take derivatives along the time dimension, the first term in the equation will have multiple multiplications of $\boldsymbol{W}^{\top}$, and the multiplication of the same matrix will cause the result to be too large or too small. These two situations correspond to the gradient explosion and gradient vanishing phenomena in recurrent neural networks. Gradient explosion can be alleviated by techniques such as gradient clipping. Gradient clipping is controlling the norm of the gradient matrix or vector within a preset range. However, the solution to the gradient vanishing phenomenon is relatively more difficult, and this has become a key problem to be addressed in recurrent neural networks. Due to the widespread existence of the gradient vanishing phenomenon, it is difficult for the gradient of one time step to be backpropagated to many steps before, which makes it difficult for traditional recurrent neural networks to capture the dependence between data from distant time steps (i.e., long-term dependencies). To deal with the long-term dependency problem, researchers have proposed other alternative RNN structures. Two commonly used RNN structures will be introduced below.

Compared to traditional recurrent neural networks, Long Short-Term Memory (LSTM) networks have added three gate control units in each step, which are forget gate $\boldsymbol{f}^{t}$, memory gate $\boldsymbol{i}^{t}$, and output gate $\boldsymbol{o}^{t}$. At the same time, in addition to storing the hidden state $\boldsymbol{h}^{t}$, the long short-term memory network also introduces a new memory unit $\boldsymbol{c}^{t}$, as shown in Fig. 3.11.

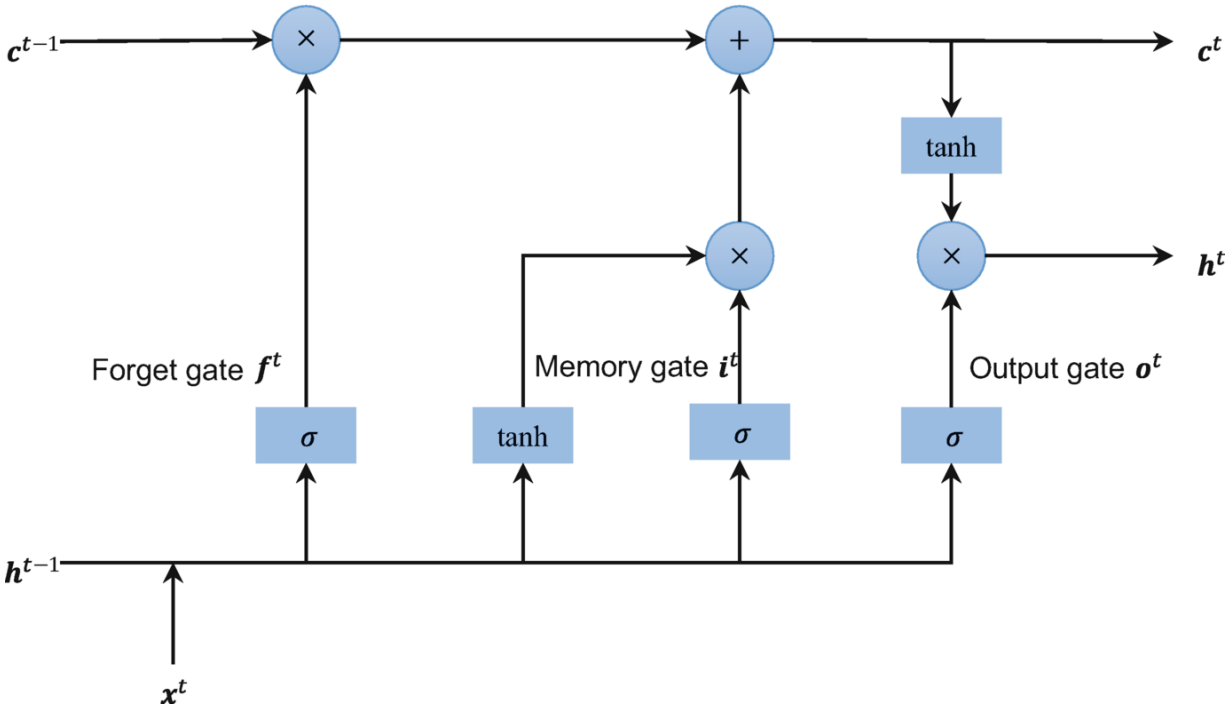


Fig. 3.11 Long Short-Term Memory (LSTM) network

The circle with the multiplication symbol $\odot$ in the figure represents the Hadamard product, which is the element-wise product of matrices. Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two matrices with the same dimensions, then the result of the Hadamard product of $\boldsymbol{C}$ is the same dimension as $\boldsymbol{A}$ and $\boldsymbol{B}$, and $C_{i,j} = A_{i,j} \times B_{i,j}$. The square with the σ in the figure represents the Sigmoid function. Specifically, the calculation process of the LSTM network is as follows:

$$\boldsymbol{f}^t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}^t] + \boldsymbol{b}_f) \tag{3.26}$$

$$\boldsymbol{i}^t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}^t] + \boldsymbol{b}_i) \tag{3.27}$$

$$\boldsymbol{o}^t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}^t] + \boldsymbol{b}_o) \tag{3.28}$$

$$\overline{\boldsymbol{c}}^t = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}^t] + \boldsymbol{b}_c) \tag{3.29}$$

$$\boldsymbol{c}^t = \boldsymbol{f}^t \odot \boldsymbol{c}^{t-1} + \boldsymbol{i}^t \odot \overline{\boldsymbol{c}}^t \tag{3.30}$$

$$\boldsymbol{h}^t = \boldsymbol{o}^t \odot \tanh(\boldsymbol{c}^t). \tag{3.31}$$

The key to LSTM alleviating the gradient vanishing problem lies in the memory unit $\boldsymbol{c}^t$. Observing the fifth equation above, there is a term $\boldsymbol{f}^t$ in the derivative of $\boldsymbol{c}^t$ with respect to $\boldsymbol{c}^{t-1}$, which is the output of a Sigmoid function. And the Sigmoid function has saturation regions on both sides. If the neural network considers some information in $\boldsymbol{c}^{t-1}$ to be important, then the corresponding $\boldsymbol{f}^t$ will be in the saturation region on the right of the Sigmoid function, and its value will be very close to 1. Even if multiple items are multiplied together, i.e., multiple steps of gradient back-propagation, it will not cause gradient vanishing.

Another alternative is Gated Recurrent Unit (GRU) which uses a similar approach to solve the problem of gradient vanishing, but it has a simpler structure and is easier to implement. It can be seen as a simplified version of LSTM networks. Considering the similarity of the memory gate and forget gate functions in LSTM, GRU combines these two gates. After the merger, GRU only has two gates, called the update gate $\boldsymbol{z}^t$ and reset gate $\boldsymbol{r}^t$. At the same time, GRU also removed the memory unit $\boldsymbol{c}^t$ in LSTM, making the hidden state $\boldsymbol{h}^t$ responsible for the memory unit at the same time. Its specific update mode is shown in Fig. 3.12.

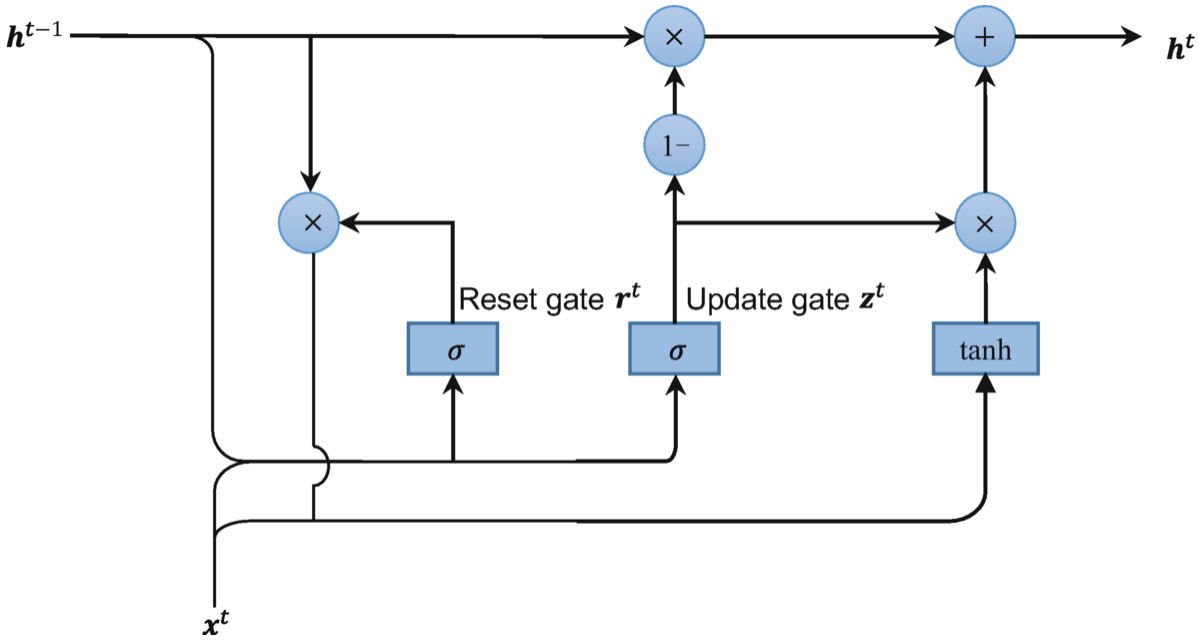


Fig. 3.12 Gated Recurrent Unit (GRU) network

The calculation of GRU is as follows:

$$\boldsymbol{z}^t = \sigma(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}^t] + \boldsymbol{b}_z) \tag{3.32}$$

$$\boldsymbol{r}^t = \sigma(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}^t] + \boldsymbol{b}_r) \tag{3.33}$$

$$\tilde{\boldsymbol{h}}^t = \tanh(\boldsymbol{W}_h \boldsymbol{x}^t + \boldsymbol{U}_h(\boldsymbol{r}^t \odot \boldsymbol{h}^{t-1}) + \boldsymbol{b}_h) \tag{3.34}$$

$$\boldsymbol{h}^t = \boldsymbol{z}^t \odot \tilde{\boldsymbol{h}}^t + (\boldsymbol{1} - \boldsymbol{z}^t) \odot \boldsymbol{h}^{t-1}. \tag{3.35}$$

Using the example of the recommendation scenario in an e-commerce platform, when performing the current time step's recommendation, the user may have previously viewed a variety of products, so the e-commerce platform's recommendation strategy needs to be based on the user's browsing history (a sequence) for the current recommendation. Since the number of items viewed by different users is not necessarily the same, the e-commerce platform's recommendation strategy needs to handle variable-length sequences, so traditional fully connected neural networks are unable to handle this problem, which makes recurrent neural networks widely used in various tasks in the recommendation scene.

3.3.3 Attention Mechanism

Attention is a commonly used mechanism in the field of deep learning, which is used to automatically learn the contribution of input data to output. In some scenarios, the input data contains a lot of irrelevant information. Attention mechanism is used to distinguish the importance of each feature in the input data and then perform subsequent tasks based on the important features. This often leads to better performance if tuned well.

In short, an attention module maps a query and a set of key–value pairs to an output, where the output is the weighted sum of the input values and the corresponding weights are calculated by the key and query. The query, key, and value are denoted as $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$, respectively. Then, an attention mechanism module can be represented as

$$\text{Output} = \boldsymbol{F}(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V}, \tag{3.36}$$

where $\boldsymbol{F}(\boldsymbol{Q}, \boldsymbol{K})$ represents the calculation of the corresponding weights based on the query and key and can usually be represented as $\boldsymbol{F}(\boldsymbol{Q}, \boldsymbol{K}) = \text{Softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)$.

The attention mechanism was first proposed in machine translation, where the translation model is composed of an encoder and a decoder which constitute a sequence-to-sequence (Seq2Seq) model. The encoder and the decoder are both composed of RNNs. When the sequence-to-sequence model is running, the encoder first receives the input data and encodes it into a vector $\boldsymbol{c}$. The decoder then decodes the vector as the initial hidden vector $\boldsymbol{h}^0 = \boldsymbol{c}$. In order to better capture the dependency relationship between the original text and the translated text, this method adds a layer of attention mechanism between the encoder and the decoder, as shown in Fig. 3.13. In the figure, the attention weight $\boldsymbol{\alpha}^{t,i}$ is calculated based on the query $\boldsymbol{h}_d^{t-1}$ and the key $(\overleftarrow{\boldsymbol{h}_e^i}, \overrightarrow{\boldsymbol{h}_e^i})$, specifically by a neural network with a final Softmax function. After the weight is calculated, the weighted sum $\boldsymbol{s}^t = \sum_{i=1}^{T} \boldsymbol{\alpha}^{t,i}[\overleftarrow{\boldsymbol{h}_e^i}, \overrightarrow{\boldsymbol{h}_e^i}]$ can be calculated for the hidden states. Then, based on the previous hidden state $\boldsymbol{h}_d^{t-1}$, the current input $\boldsymbol{x}^t$, and the weighted sum $\boldsymbol{s}^t$, the prediction of the current time step can be derived.

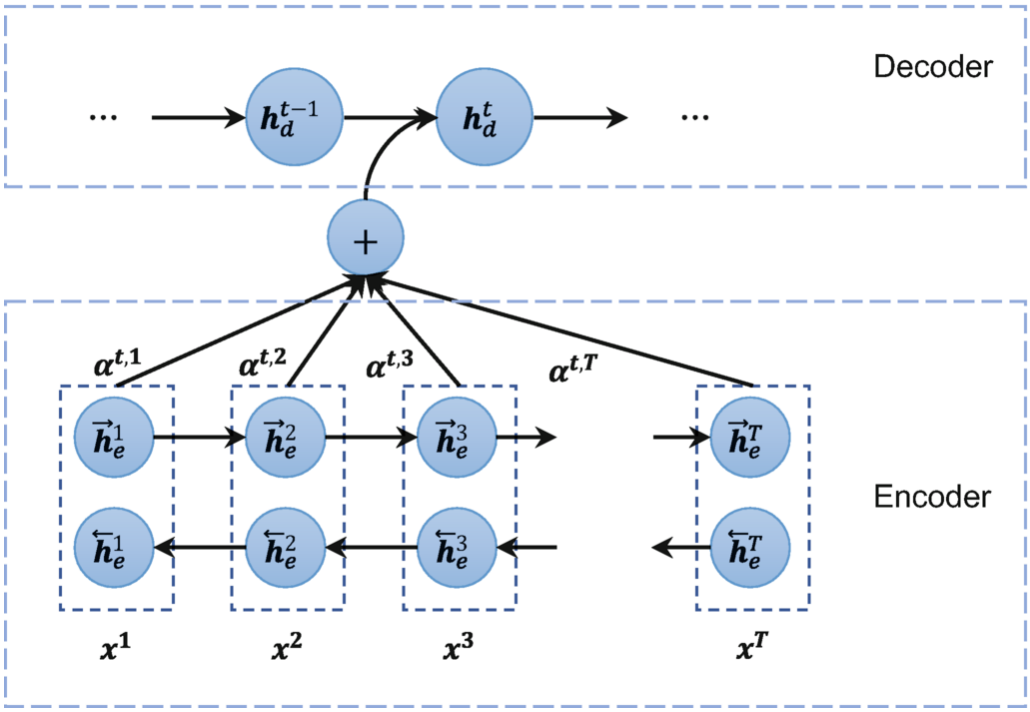


Fig. 3.13 The attention mechanism in machine translation

It is worth noting that if the key and value are the same content, both being $(\overleftarrow{\boldsymbol{h}}_e^i, \overrightarrow{\boldsymbol{h}}_e^i)$, it is called a self-attention mechanism. Recently, the self-attention mechanism has received widespread attention. At its core, its logic is consistent with traditional attention mechanisms. The difference is that self-attention mechanisms operate on attention mechanisms between the same set of data, while traditional attention mechanisms operate on attention mechanisms between two sets of data. For example, in Fig. 3.13, it is the operation of attention mechanisms between the decoder's hidden state and the encoder's hidden state. At the same time, in a self-attention mechanism, related research has also proposed several special structures of attention mechanisms, namely, scaling dot product attention and multi-head attention, the specific structure is shown in Fig. 3.14. Among them, scaling dot product attention is actually a way of calculating weights, that is, $\boldsymbol{F} = \mathrm{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)$, where d_k represents the dimension of the query. The multi-head attention mechanism also uses multiple different attention mechanism modules (different parameters and similar structures) and finally concatenates the outputs of different attention modules to better mine information in the data and use it for subsequent tasks.

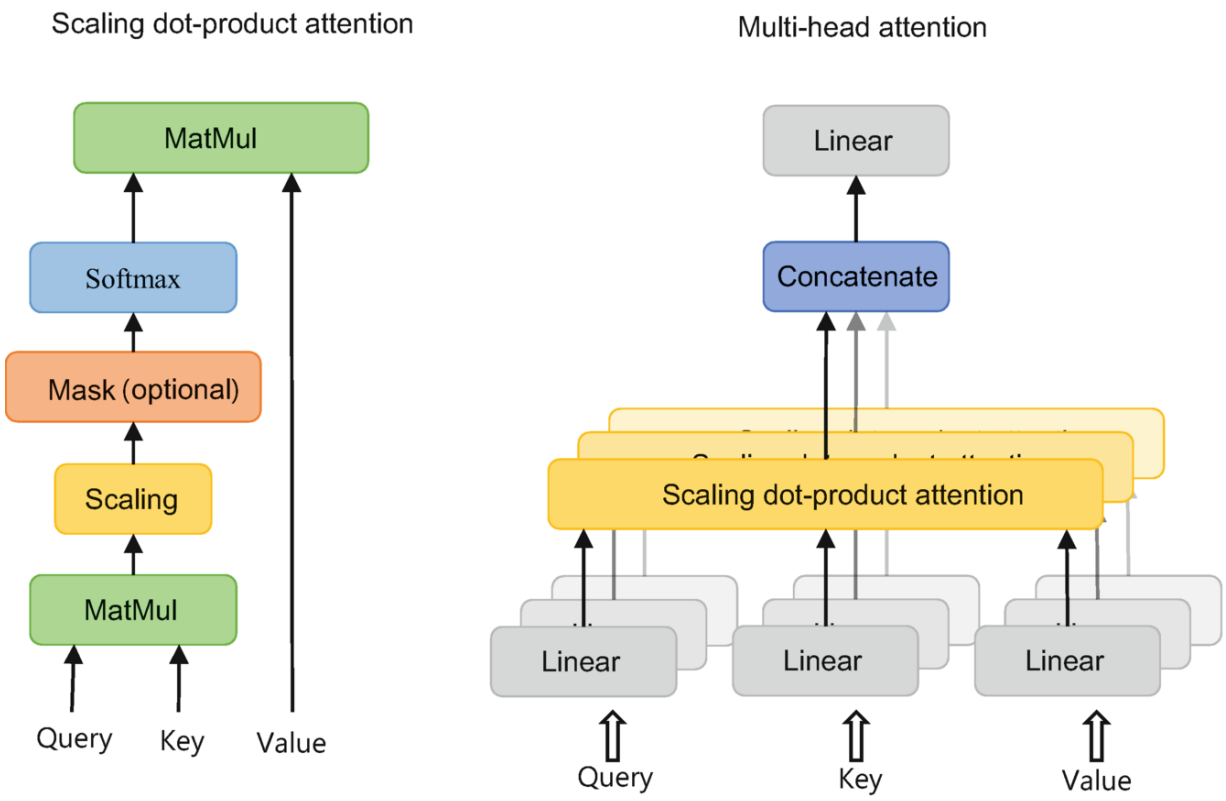


Fig. 3.14 Scaling dot product attention (left) and multi-head attention (right)

This section still uses the recommendation scenario of the e-commerce platform as an example to show the application of the attention mechanism. In order to better complete the recommendation task, it is necessary to accurately model the user's interests based on the user's browsing records. However, due to random browsing behavior by the user during the activities, that is, noise in the user's browsing history, it may not be possible to obtain the best results by directly using all the data. Moreover, the user's browsing history may be very long, and even long-term memory networks are difficult to effectively encode. Additionally, there is a long-term dependence between the user's behaviors, so it is also unreasonable to model user's interests with only the most recent browsing history. Therefore, researchers propose using an attention mechanism to model the user's interests [5]. Given the current item, the algorithm will perform an attention mechanism operation between the item and the user's historical browsing records and then judge the current user's preference for the item based on the output of the attention mechanism, that is, the user's interests. This specific process is shown in Fig. 3.15. Thanks to the use of the attention mechanism, the model significantly improved the performance of the recommendation system online.

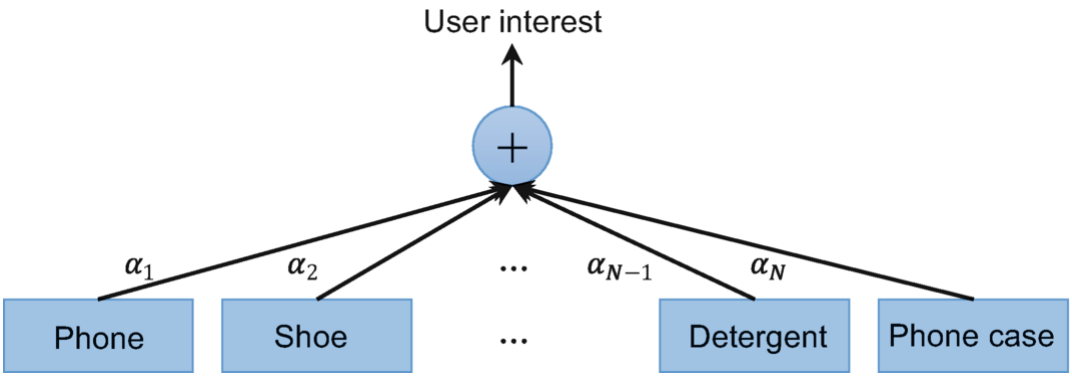


Fig. 3.15 Modeling user interests

3.3.4 Sequence Modeling and Pre-training

In recommendation scenarios, the input data of most tasks are usually presented in the form of sequence, so people can view the items that can be recommended as one word and view the user's behavior as a sentence composed of word sequences. Then, it is natural to adopt the sequence modeling algorithm that is at the forefront of natural language processing. On the other hand, in scenarios such as news recommendations, the input data is presented in the form of natural language. Given the relevance of the recommendation field and the natural language processing field, this section will introduce three sequence modeling and pre-training techniques: Word2Vec, Transformer, and BERT from the natural language processing field.

Word2Vec

The Word2Vec model [2] aims to learn representations of words. Essentially, it utilizes the hidden layer of a neural network to achieve the distributional representation of discrete data in the hidden space, mapping words from a discrete space to a multi-dimensional real-valued hidden space. The input and output of Word2Vec are both one-hot encoded vocabulary vectors. It trains on all the data in a large-scale natural language corpus and, after convergence, the vector from the input layer to the hidden layer is the corresponding word's distributed representation, i.e., the word vector.

Word2Vec uses an unsupervised training mode and has two models, CBOW and Skip-Gram. CBOW is relatively more suitable for smaller datasets, while Skip-Gram performs better in relatively larger corpus. The CBOW model is shown in Fig. 3.16a, it uses the context (neighboring

words in the corpus) of the target word as input, and after weighting sum processing in the mapping layer, it optimizes for the correct output of the target word. On the other hand, the Skip-Gram model takes the current word as input and aims to correctly predict the context words, as shown in Fig. 3.16b.

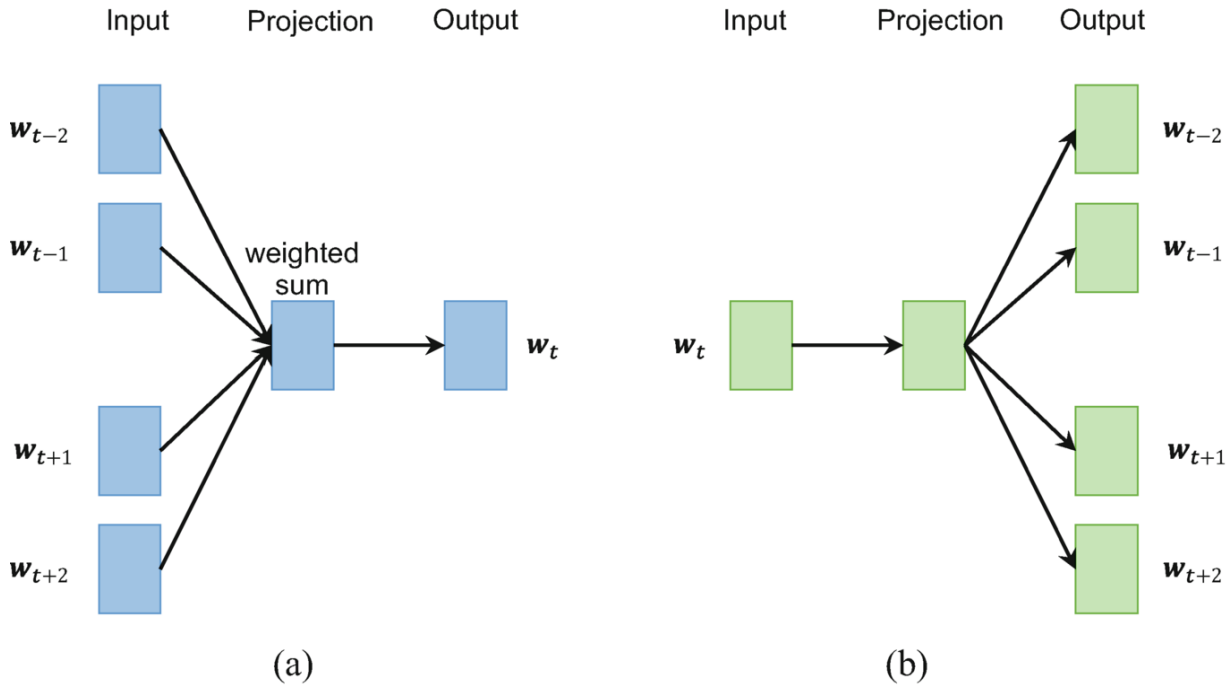


Fig. 3.16 Word2Vec model. (**a**) CBOW model. (**b**) Skip-Gram model

Transformer

Transformer, as described in [4], uses a fully connected network with self-attention mechanisms and position embeddings to replace recurrent neural networks. This breakthrough removed the limitation of serial computation based on the input's temporal sequence in RNNs, and its derived pre-trained text representation model BERT has excelled in various downstream tasks, becoming one of the mainstream frameworks in natural language processing.

As shown in Fig. 3.17, Transformer mainly consists of an encoder and a decoder, corresponding to the upstream and downstream tasks, respectively: the upstream task trains a text representation model, while the downstream task performs specific tasks such as classification and text generation.

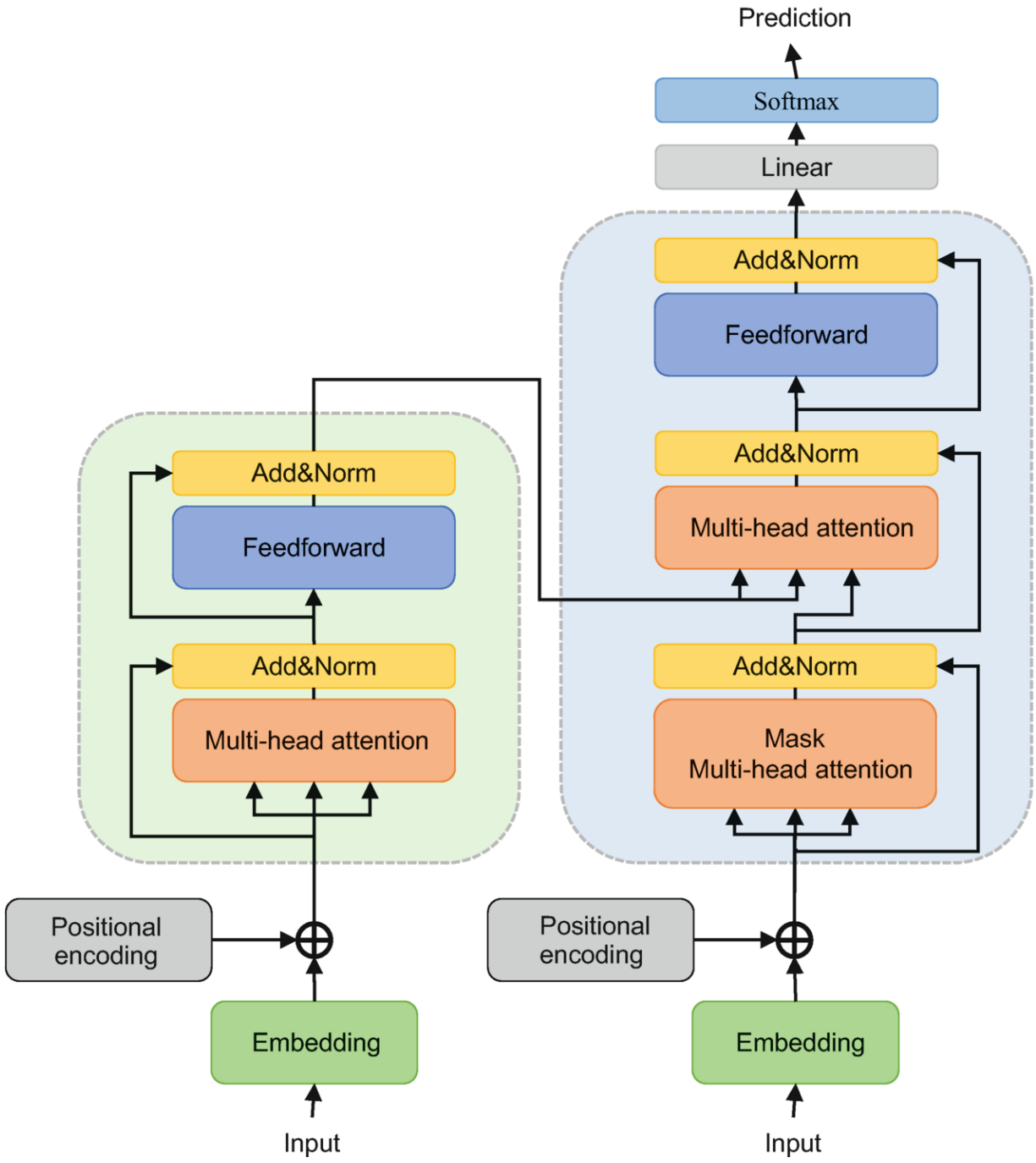


Fig. 3.17 Transformer model

During execution, Transformer first initializes the input representation by encoding the text data into an initial representation. Generally speaking, any method mentioned earlier in this text can be used here. Additionally, since Transformer uses a fully connected network, discarding the sentence's sequential information, it is necessary to add its position encoding to the input data in order to consider the data's sequential information. In the

position encoding section, trigonometric functions are used to superimpose the position information of the words in the whole sentence. The formula is as follows:

$$\mathrm{PE}_{\mathrm{pos},2i} = \sin(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}) \tag{3.37}$$

$$\mathrm{PE}_{\mathrm{pos},2i+1} = \cos(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}). \tag{3.38}$$

In the formula, “pos” represents the position of the word in the entire sentence, i represents the dimension index of the word vector, and d_{model} represents the dimension hidden vector in the attention mechanism. For the dimension *i* of the word vector, the encoding value increases with the position index, showing a triangular wave fluctuation, and the fluctuation period increases exponentially with the dimension index *i*.

The core of Transformer is the self-attention mechanism described earlier. Its encoder and decoder modules mainly utilize multi-head self-attention, which independently carry out self-attention operations on multiple sets of queries, keys, and values and then combine the extracted information.

In Transformer, in order to avoid the temporal dependency problem brought by recurrent neural networks and speed up the model training, it removed the serial computation method of recurrent neural networks and directly used fully connected layers as the main modules. As a result, it can independently calculate the attention between all words in the sentence. Through this mechanism, Transformer can better handle long-term dependency problems. On the other hand, this mechanism also overlooks the sequence information of the sentence, but the incorporation of position encoding made up for this. The combination of self-attention mechanism and position embedding is the ingenuity of Transformer’s design.

BERT

BERT [1] is one of the most popular frameworks in the field of natural language processing currently. BERT is essentially the encoder part of the Transformer, used to generate a text representation of a sentence for downstream tasks. The training of BERT is an unsupervised process, which can be achieved through the use of masked language model (MLM) and next sentence prediction (NSP) pre-training tasks.

MLM first randomly masks or replaces words in a sentence, and then the downstream model predicts the covered or replaced words through context and finally constructs a loss function that is only for the prediction part to train the BERT model. To prevent overfitting and improve the model's understanding of the text itself, MLM uses a mixed method when covering or replacing words, with most (80%) words being covered as "[mask]" a small portion (10%) being replaced with other words, and a small portion (10%) remaining unchanged.

When the task requires the sentence-level representation, MLM tends to extract word-level representations, and then the NSP task needs to be pre-trained. The goal of the NSP task is to predict whether two sentences are connected. Specifically, NSP takes *N* pairs of sentences with a 50% connected probability from the corpus, adds [cls] prediction tags and [sep] sentence tags, inputs them into the BERT model, uses the global representation collected by the [cls] prediction tag to perform binary classification prediction, and optimizes the BERT model using the classification loss.

MLM and NSP tasks can be performed simultaneously as shown in Fig. 3.18. Both tasks require data from unlabeled text data and are self-supervised. This greatly reduces the data cost and, combined with the parallel training of the Transformer, BERT can be trained on a large-scale corpus, providing high-quality and transferable pre-trained text representations for downstream tasks.

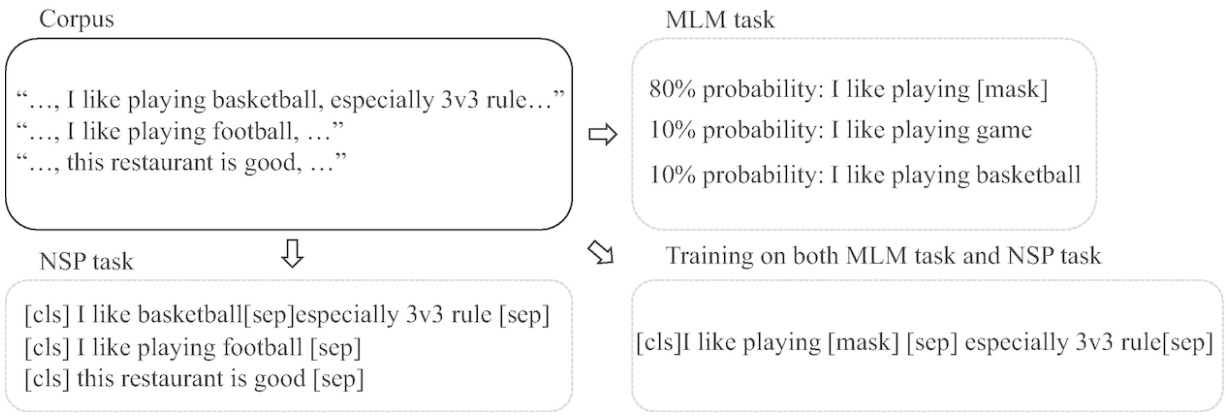


Fig. 3.18 BERT training paradigm

3.4 Conclusion

This chapter introduces the basics of deep learning, including feedforward computation and back-propagation algorithms for deep neural networks, as well as various classic neural network models. As readers learn, they can combine the content of other chapters in this book to understand and design different types of neural network models for recommendation scenarios, taking into account the data characteristics and task properties, in order to improve recommendation performance.

References

1. Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.
2. Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. https://doi.org/10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.
3. David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. https://doi.org/10.1038/323533a0.
4. Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30. 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, California, USA: Curran Associates, Inc., 2017, pp. 6000–6010. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
5. Guorui Zhou et al. “Deep Interest Network for Click-Through Rate Prediction”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 1059–1068. ISBN: 9781450355520. https://doi.org/10.1145/3219819.3219823.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_4

4. Deep Learning-Based Recommendation Algorithms

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter introduces the relationship between collaborative filtering and deep learning and then presented various deep learning-based collaborative filtering algorithms. Leveraging cutting-edge methods from deep learning, these algorithms can significantly improve the accuracy, scalability, diversity, and interpretability of recommendation systems, offering richer technological choices for recommendation system design. However, most of these algorithms are optimized for specific problems, and there are often limitations in practical applications. Therefore, at the system design level, algorithm integration or fusion needs to be considered.

Keywords Deep learning-based collaborative filtering

Deep learning technology has been changing the progress of AI. Deep learning has also shed significant impact on the recommender system research. On one hand, the researchers can make use of the deep learning technology to enhance the modeling performance of the conventional

recommendation algorithms. On the other hand, there are novel recommendation algorithms being developed that are inspired by the deep learning technology. The deep learning-based algorithms are not merely heated in the academia, they are also widely applied in the industry. The deep learning-based algorithms have the strong capability for representation and generalization, so that they can enhance the conventional algorithms in many ways. In this chapter, the six of the most important topics in the deep learning -based recommendation algorithms are reviewed, which are collaborative filtering, feature product, graph learning, sequential recommendation, knowledge distillation, and deep reinforcement learning.

4.1 Deep Learning and Collaborative Filtering

Collaborative filtering is one of the most classic ideas in recommender algorithms. The collaborative filtering algorithm does not need to collect the feature or content information about users or items; instead, it creates the model based on the user–item interactions, from which it learns the similar users or items based on the interaction behavior and produces the recommendations that are similar to what users prefer. Usually, there are two types of implementations of the collaborative filtering algorithms: the matrix factorization model-based one and the memory-based one. Along with the development of the deep learning technology, it has been found that the matrix factorization model stays on the surface and its expressibility can be further improved. In this subsection, the evolution of the collaborative filtering algorithms under the impact of deep learning is discussed.

4.1.1 Restricted Boltzmann Machine-Based Collaborative Filtering

In the contest of “Netflix Prize” in 2006, there were two algorithms receiving great attention due to its elegant mathematical theory and completed experimental proof. One of the two algorithms is Singular Value Decomposition, and another is Restricted Boltzmann Machine, i.e., RBM. RBM is a generative random neural network. Ruslan Salakhutdinov et al. [49] adjusted it to be a collaborative filtering algorithm and published it in the proceedings of ICML 2007. The architecture of the RBM model is shown in Fig. 4.1. It mainly consists of a hidden layer, a visible layer, and model parameters. The visible layer takes the seen data of users as input.

Every node represents an item, and its content is one-hot coded. Using the movie recommendation as an example, the ratings that the user gives to each corresponding movie are encoded into the one-hot code, while those movies that do not have ratings are given missing values. In the hidden layer, each of the neurons is a binary unit, which has merely two states, that is, activated (1) and deactivated (0). This state is used to represent the principle that for each of the users, the units in the hidden layer are connected to the non-missing nodes in the visible layer, but the internal nodes inside the hidden layer are not. In addition, there is a bias parameter for each node. To differentiate between the bias parameter in the hidden layer and the visible layer, the two variables are represented as a and b, respectively.

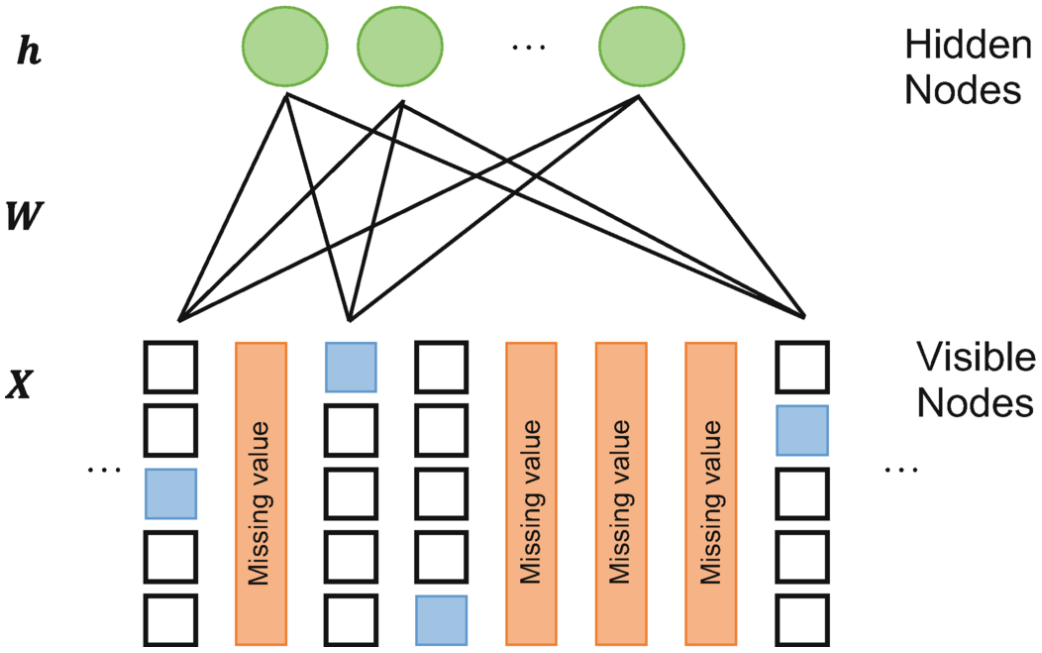


Fig. 4.1 Architecture of the restricted Boltzmann Machine

RBM uses a multinomial distribution to model the ratings that the users have given to the items. The probability of the multinomial distribution is obtained by performing a normalization by using a Softmax function on the predicted ratings.

$$p(\boldsymbol{x}_i^k = 1 \mid \boldsymbol{h}) = \frac{\exp\left(\boldsymbol{b}_i^k + \sum_{j=1}^{F} \boldsymbol{h}_j \boldsymbol{W}_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(\boldsymbol{b}_i^l + \sum_{j=1}^{F} \boldsymbol{h}_j \boldsymbol{W}_{ij}^l\right)}. \tag{4.1}$$

The probability of activation of each hidden unit is

$$p(\boldsymbol{h}_j = 1 \mid \boldsymbol{X}) = \sigma\left(\boldsymbol{a}_j + \sum_{i=1}^{m}\sum_{k=1}^{K} \boldsymbol{x}_i^k \boldsymbol{W}_{ij}^k\right). \tag{4.2}$$

σ in the above equation is the Sigmoid function. The energy function is shown below.

$$E(\boldsymbol{X}, \boldsymbol{h}) = -\sum_{i=1}^{m}\sum_{j=1}^{F}\sum_{k=1}^{K} \boldsymbol{W}_{ij}^k \boldsymbol{h}_j \boldsymbol{x}_i^k - \sum_{i=1}^{m}\sum_{k=1}^{F} \boldsymbol{x}_i^k \boldsymbol{b}_i^k - \sum_{j=1}^{F} \boldsymbol{h}_j \boldsymbol{a}_j. \tag{4.3}$$

The marginal distribution is shown below.

$$p(\boldsymbol{X}) = \sum_{\boldsymbol{h}} \frac{\exp(-E(\boldsymbol{X}, \boldsymbol{h}))}{\sum_{\boldsymbol{X}', \boldsymbol{h}'} \exp(-E(\boldsymbol{X}', \boldsymbol{h}'))}. \tag{4.4}$$

In theory, the optimization of the RBM model maximizes the probability of the marginal distribution. The derivatives of the distribution include one step that needs to iteratively calculate all the possible values to compute the integral, and this generates tremendous overhead in computation. Hence, Ruslan Salakhutdinov et al. proposed using the contrastive divergence (CD) [24] to accelerate the process. The enhanced steps that update the parameters are shown below.

$$\Delta \boldsymbol{W}_{ij}^k = \varepsilon\left(< \boldsymbol{x}_i^k \boldsymbol{h}_j >_{\text{data}} - < \boldsymbol{x}_i^k \boldsymbol{h}_j >_T\right) \tag{4.5}$$

$$\Delta \boldsymbol{b}_i^k = \varepsilon\left(< \boldsymbol{v}_i^k >_{\text{data}} - < \boldsymbol{v}_i^k >_T\right) \tag{4.6}$$

$$\Delta \boldsymbol{a}_j = \varepsilon\left(< \boldsymbol{h}_j >_{\text{data}} - < \boldsymbol{h}_j >_T\right). \tag{4.7}$$

The parameter $< \cdot >_{\text{data}}$ in the above formula indicates the co-occurrence in the training set. $< \cdot T$ indicates the co-occurrence after T times CD sampling. The computation of CD is rather simple. It is essentially a Gibbs sampling process. Based on (4.2), the activation probability of the hidden vectors can be obtained, and then a Bernoulli experiment is conducted to confirm whether the unit is activated or not. Then, by using (4.1), the activation probabilities of the non-missing nodes in the visible layer can be determined by sampling from the multinomial distribution. This procedure is repeated for T times. Usually, the value of T can be small,

e.g., 1, to get a good result. In the last, using the final results after the T times iteration to update the model parameters.

4.1.2 Autoencoder-Based Collaborative Filtering

In fact, the autoencoder (AE) model is very similar to RBM. That is, the computation from the visible layer to the hidden layer can be regarded as the encoding process that compresses information, while the computation from the hidden layer to the visible layer can be treated as the decoding process. The biggest difference between RBM and AE is that RBM is a probabilistic generative neural network so that its computation relies on the Gibbs sampling instead of the end-to-end gradient descent approach. AE is a deterministic neural network. Its training process is rather simple and efficient, owing to which it is preferred in many applications. Suvash Sedhain [50] proposed the CF algorithm that is based on the AE model, termed AutoRec, and the method demonstrated superior performance on the Movielens and Netflix dataset compared to RBM. The model structure of the AutoRec algorithm is illustrated in Fig. 4.2. It consists of a neural network that includes a hidden layer. The input is a long vector, which represents the ratings on all the items from a particular user, i.e., $\boldsymbol{r}^{(i)} = (\boldsymbol{R}_{i1}, \boldsymbol{R}_{i2}, \boldsymbol{R}_{i3}, \cdots, \boldsymbol{R}_{im})$ $\boldsymbol{r}^{(i)}$. The non-missing values in the vector are compressed into a low-dimensional latent vector after the hidden layer. This vector indicates the interests of the users in the low-level latent space, with which the users' ratings can be reconstructed.

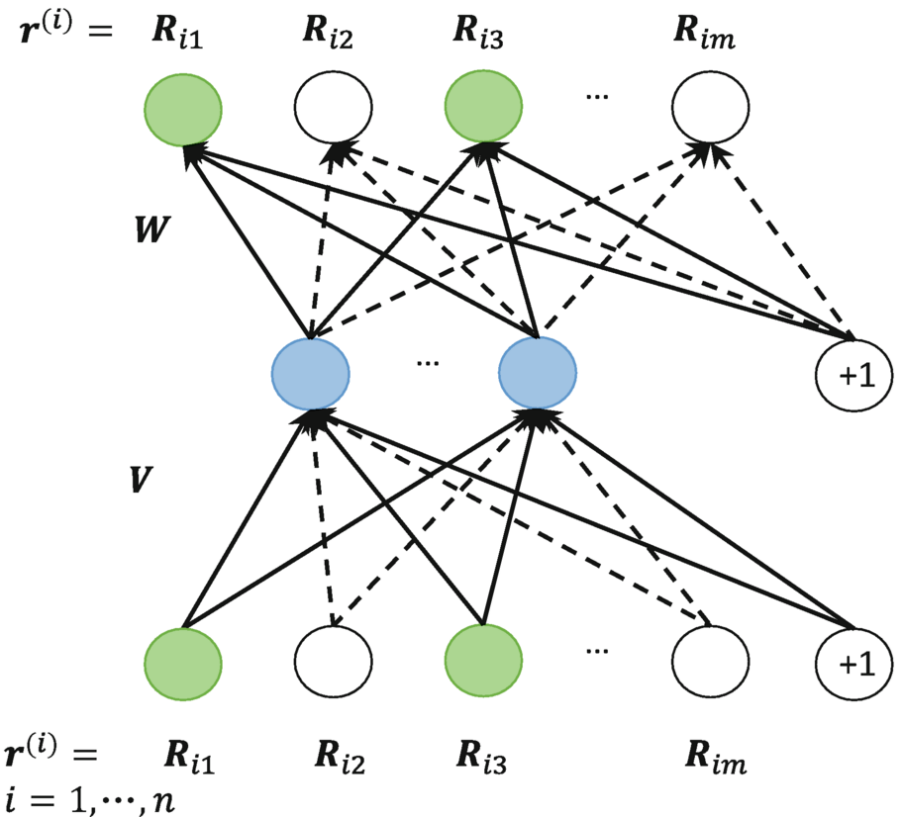


Fig. 4.2 The user-based AutoRec model

The entire flow can be formalized by using the following equation:

$$\hat{\boldsymbol{r}}^{(i)} = f\left(\boldsymbol{W} \cdot g\left(\boldsymbol{V}\boldsymbol{r}^{(i)} + \boldsymbol{b}_v\right) + \boldsymbol{b}_w\right). \tag{4.8}$$

In the above equation, $f(\cdot)$ and $g(\cdot)$ are the optional activation functions. To introduce non-linearity into the model, the activation functions such as Sigmoid, tanh, etc., can be properly chosen based on the actual dataset. The optimization objective of the AutoRec model is to minimize the RMSE score computed between the reconstructed ratings and the original ground truth. The training process can be performed by using the gradient descent method. The model is called user-based AutoRec due to the fact that the input vector is the user-related data. Similarly, the input vector can be replaced by using item-based rating data, that is, $\boldsymbol{r}^{(i)} = (\boldsymbol{R}_{1i}, \boldsymbol{R}_{2i}, \boldsymbol{R}_{3i}, \cdots, \boldsymbol{R}_{ni})$. Correspondingly, the model is called the item-based AutoRec.

The training process of AutoRec can be self-supervised. In the ideal situation, the model is expected to reconstruct the input vectors, and this is beneficial to the application scenarios of information compression. In the

context of recommendation, many times the ratings of users are extremely sparse, and this leads to overfitting of AutoRec, i.e., it can well reconstruct the historical ratings but cannot predict precisely the unknown ratings. Therefore, the Denoising Autoencoders (DAEs) are applied in the recommendation use case. That is, the input vectors are either denoised or noised to allow the decoder to reconstruct the original correct values. By doing this, the latent vectors after the encoding process do well in generalization. The classic approach for this method is the one called CDAE that was proposed by Yao Wu in [72].

Both AutoRec and CDAE deterministically encode the input vectors into the latent vectors. Variational AutoEncoder (VAE) is yet another autoencoder. It is special in a way that the encoder is a generative model. VAE assumes that the latent vector of the input vector is not deterministic. Instead, it follows the normal distribution, i.e., $N(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$. The encoder draws the mean $\boldsymbol{\mu}$ and the standard deviation $\boldsymbol{\sigma}$ from the distribution and then samples the latent vectors that are then fed into the encoder to generate the model. By using the generative model in the encoder, VAE becomes more flexible in the expressibility compared to the conventional approach. In addition, it also adds diversity to the recommendation results. The model structure of VAE is shown in Fig. 4.3. Inspired by the idea of VAE, Dawen Liang et al. proposed the novel collaborative filtering algorithm, named Multi-VAE [36], which improves the model performance of the original VAE by using the multinomial likelihood function that demonstrates superiority over those of the Gaussian distribution and Logistic distribution. Given that the latent vectors are drawn from $N(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, the sampling process is non-differentiable. This makes it infeasible to optimize the VAE model in an end-to-end flow like the other normal AE models. To address this issue, the reparameterization trick is introduced. That is, from $N(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, a vector $\boldsymbol{h}$ is sampled, which is equivalent to sampling a vector $\boldsymbol{\varepsilon}$ from $N(\mathbf{0}, \boldsymbol{I}^2)$. The sampled vector is then converted into a desired one $\boldsymbol{h} = \boldsymbol{\mu} + \boldsymbol{\varepsilon} \cdot \boldsymbol{\sigma}$. As a result, the sampling process for the vector $\boldsymbol{\varepsilon}$ does not require differentiation, and the objective function can still use the variables of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ for back-propagation for the encoder. In addition to the bias in the reconstruction function of VAE, there is another loss function of KL divergence, which is used as constraints to guarantee that the latent vectors follow the standard distribution. Here, the computation results are shown directly.

$$L_{\boldsymbol{\mu},\boldsymbol{\sigma}^2} = \frac{1}{2}\sum_{i=1}^{d} \boldsymbol{\mu}_{(i)}^2 + \boldsymbol{\sigma}_{(i)}^2 - \log \boldsymbol{\sigma}_{(i)}^2 - 1. \tag{4.9}$$

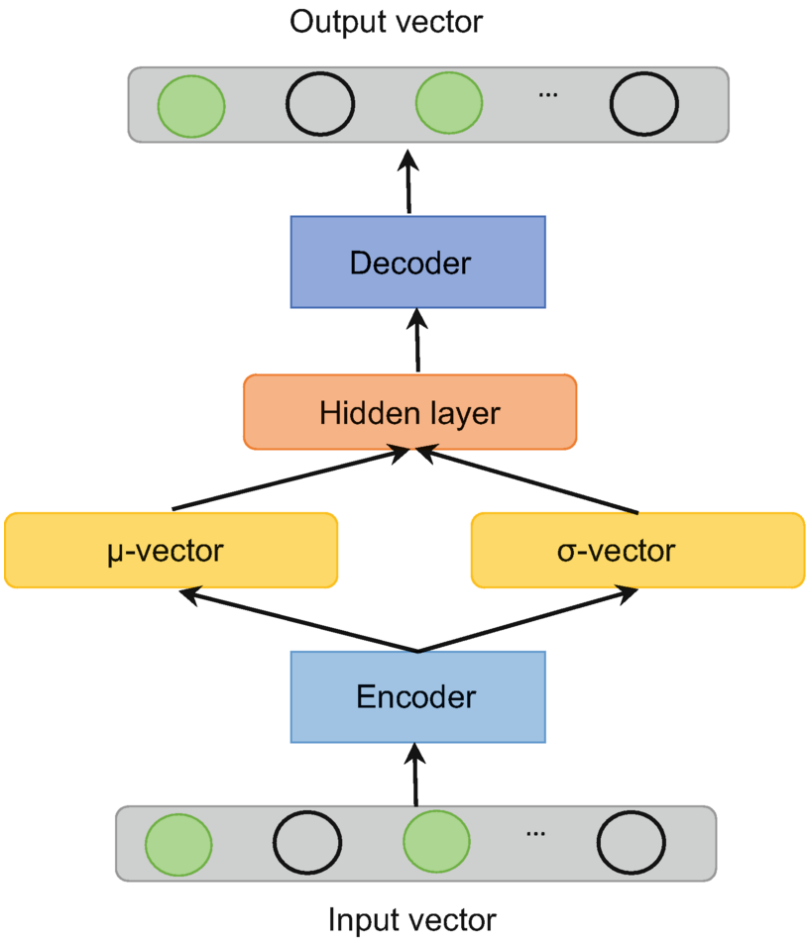


Fig. 4.3 Model structure of VAE

In the above equation, $\boldsymbol{\mu}_{(i)}$ is used to represent the *i*-th dimension of the vector $\boldsymbol{\mu}$.

4.1.3 Deep Learning and Matrix Factorization

Neural Collaborative Filtering

Xiangnan He et al. [22] pointed out that in the conventional matrix factorization method that is used for collaborative filtering, the interaction between user and item can only be modeled by using the dot product of the latent vectors, and this restricts the expressibility of the model. Considering the potential of the neural network that any form of the functions can be fitted, Xiangnan He et al. proposed the Neural Matrix Factorization

(NeuMF), where the multi-layer perception is used to improve the modeling and generalizing capabilities for non-linearities. The model structure of NeuMF is shown in Fig. 4.4. It consists of two branches. The left one is the generalized version of the conventional matrix factorization, called Generalized Matrix Factorization (GMF). GMF is designed to differentiate the importance of the latent vectors at different dimensions. Therefore, it uses a linear regression layer to fuse the user vector and the item vector by doing the dot product. The right part of the NeuMF model is a multi-layer perception (MLP) layer. Differently, it does not perform the element-wise product of the vectors. Instead, it concatenates the vectors for users and items that are used as input to the MLP, to learn the user–item interactions. The output from GMF and MLP is processed in a linear confusion function and then a Sigmoid activate function to get the final score. The entire model is optimized based on the binary cross-entropy metric. There are two details that are worth mentioning. One is that the GMF part and the MLP part do not share the latent vectors for users or items—they have the latent vectors for each separately—this design makes the model perform better and more flexible compared to the single approach of each. This is because that the dimensionality of the latent vectors in the two parts may not be identical. The other one is that both of the GMF part and the MLP part can be pre-trained and used for initializing the corresponding modules in the NeuMF model, and by doing this, the local optimization can be avoided.

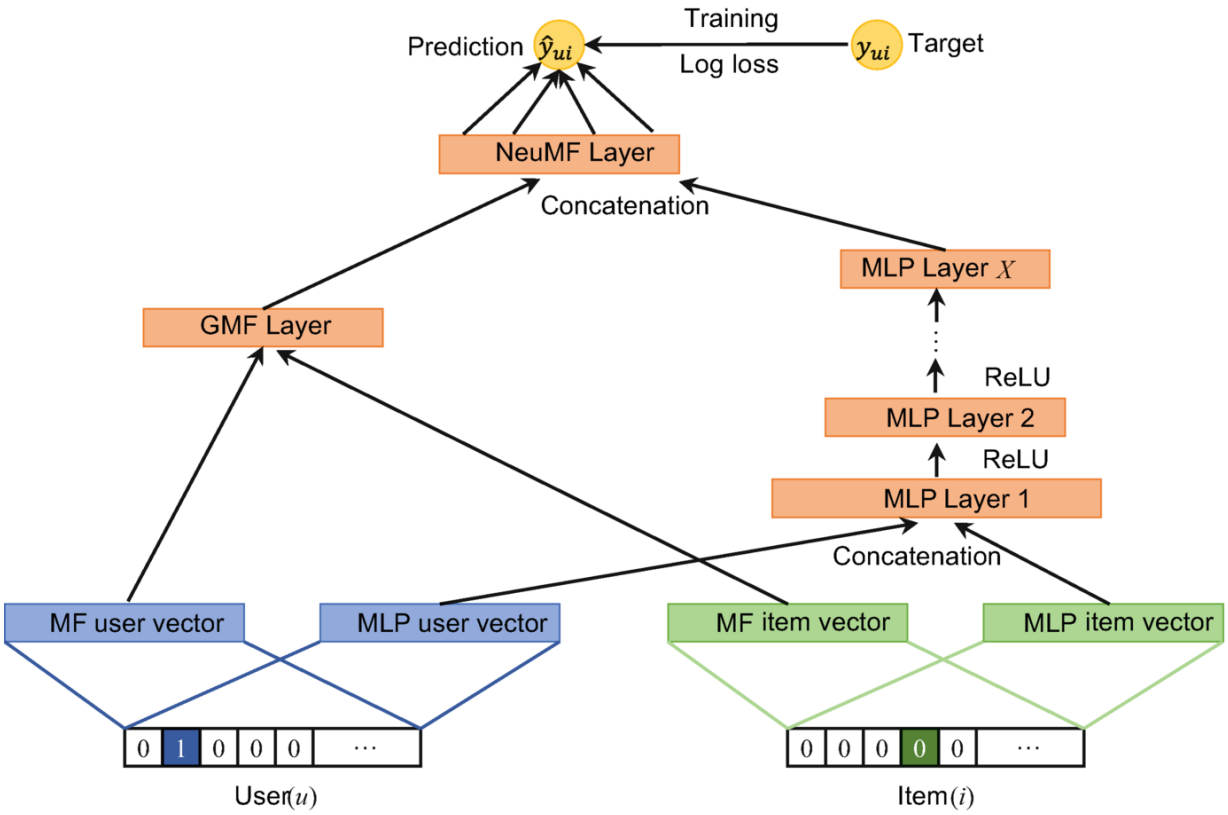


Fig. 4.4 Model structure of the NeuMF

The proposed NeuMF creates a widely spread interest in the Neural Collaborative Filtering method from the researchers in the field. Nowadays, it has become one of the baseline methods in the research of the collaborative filtering algorithms. Interestingly, there have been many research articles published after the idea of Neural Collaborative Filtering was developed. Though these approaches had similar structures, they provided new inspirations to the following research in the realm. For example, Zhi-Hong Deng et al. in the DeepCF [12] proposed that the collaborative filtering algorithms that are based on the intrinsic latent vectors can be categorized into two groups. One aims at mapping the user and item representations into a low-dimension space and leverages the relationship among the vectors (e.g., the dot product or the Cosine similarity) to indicate the users' preferences. The other one aims at learning the complex matching function, which relies on the user or item representations, respectively, without the need to align the user vectors and item vectors within the same space. Essentially, the GMF layer in NeuMF corresponds to the two different collaborative filtering paradigms.

Deep Matrix Factorization

Inspired by the modeling methods used for the correlation between documents and the search query texts in DSSM [25], Hong-Jian Xue et al. proposed the two-tower architecture based on MLP [75] and named it Deep Matrix Factorization (DMF), to improve the conventional matrix factorization algorithms. The model structure of DMF is demonstrated in Fig. 4.5. The left and right sides of the model correspond to the user module and the item module. Compared to NeuMF where the users and the items are represented in one-hot code, the inputs of DMF use all the historic ratings of a user to represent the user (the same is applied to the items). This is similar to RBM and AutoRec.

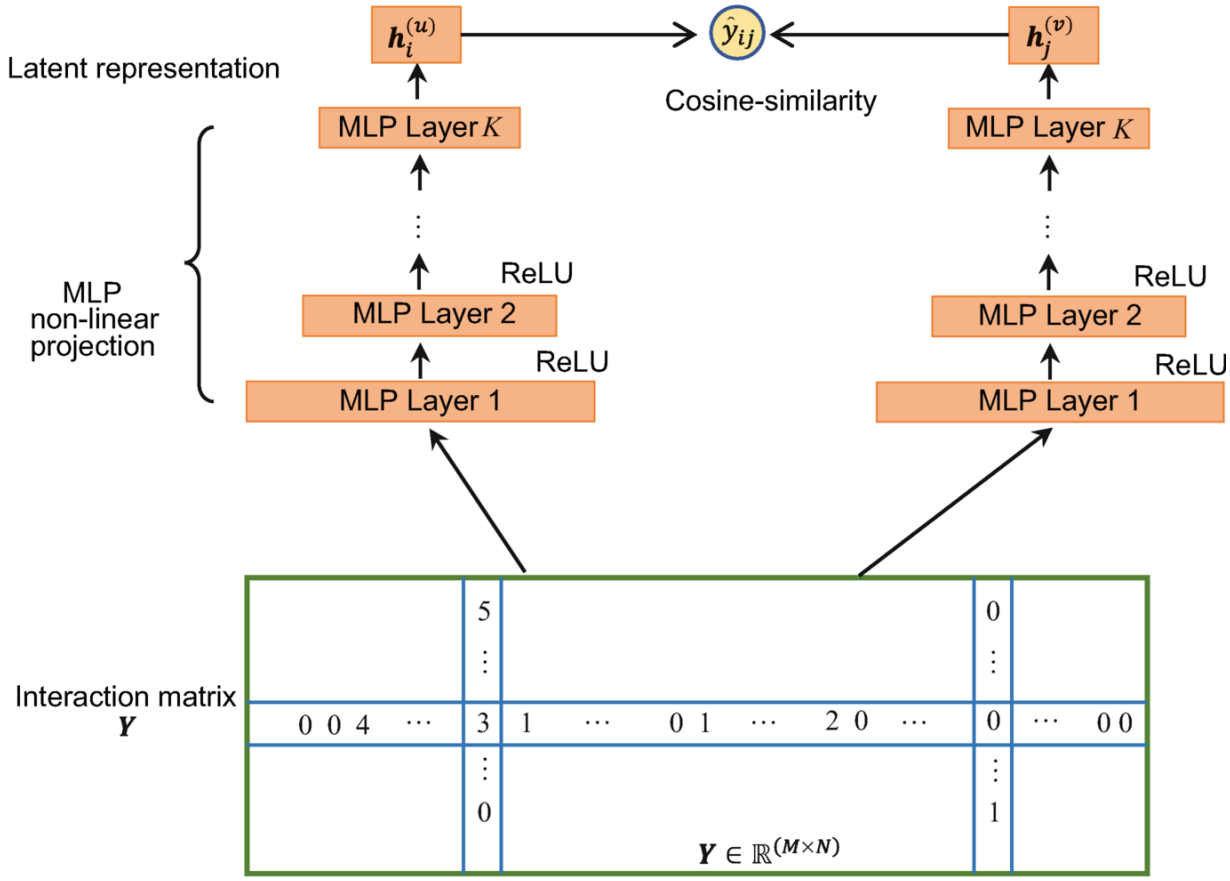


Fig. 4.5 Model structure of DMF

Assuming that the user and item rating matrix in the data is indicated as $\boldsymbol{Y} \in \mathbb{R}^{(M \times N)}$, then the input of the user side is one row of $\boldsymbol{Y}$, which is $\boldsymbol{u}_i = \boldsymbol{Y}_{i*}$, and the input of the item side is one column of the $\boldsymbol{Y}$, which is

$\boldsymbol{v}_j = \boldsymbol{Y}_{*j}$. After the transformation in MLP, the inputs can be mapped to the low-dimension representations as below.

$$\boldsymbol{l}_1 = \boldsymbol{W}_1 \boldsymbol{x} \tag{4.10}$$

$$\boldsymbol{l}_i = f(\boldsymbol{W}_{i-1}\boldsymbol{l}_{i-1} + \boldsymbol{b}_i),\ i = 2, \cdots, N-1 \tag{4.11}$$

$$\boldsymbol{h} = f(\boldsymbol{W}_N \boldsymbol{l}_{N-1} + \boldsymbol{b}_N). \tag{4.12}$$

In the above equation, $\boldsymbol{x}$ refers to the vectors of either $\boldsymbol{u}$ or $\boldsymbol{v}$. It is worth noting that the user side and item side have two different sets of the MLP parameters. Finally, DMF uses the cosine similarity to represent the rating predictions for user i on item j.

$$\hat{\boldsymbol{Y}}_{ij} = \text{cosine}(\boldsymbol{h}_i^{(u)}, \boldsymbol{h}_j^{(v)}) = \frac{\boldsymbol{h}_i^{(u)\top} \boldsymbol{h}_j^{(v)}}{\|\boldsymbol{h}_i^{(u)}\| \cdot \|\boldsymbol{h}_j^{(v)}\|}. \tag{4.13}$$

In addition, DMF changes the loss function. By taking into account that the RMSE is merely suitable to the prediction for the explicit rating scenarios, while cross entropy is merely suitable for the implicit rating scenarios, Hong-Jian Xue et al. proposed the normalized cross entropy as the loss function that works for both the explicit and implicit ratings.

$$L = -\sum_{(i,j) \in \boldsymbol{Y}^+ \cup \boldsymbol{Y}^-} \left(\frac{\boldsymbol{Y}_{ij}}{\max(\boldsymbol{Y})} \log \hat{\boldsymbol{Y}}_{ij} + \left(1 - \frac{\boldsymbol{Y}_{ij}}{\max(\boldsymbol{Y})}\right) \log(1 - \hat{\boldsymbol{Y}}_{ij}) \right) \cdot \tag{4.14}$$

In the above equation, $\boldsymbol{Y}^+$ and $\boldsymbol{Y}^-$ are the positive and negative samples. $\max(\boldsymbol{Y})$ is the maximum rating value (e.g., 5 in the scale of 1 to 5). It is not hard to observe that generally normalized cross entropy adds weighting operations on the basis of the cross entropy.

4.1.4 Neighborhood-Based Collaborative Filtering

The deep collaborative filtering algorithms that have been introduced thus far in this chapter all use the latent factor-based method. Another classic collaborative filtering method is neighborhood-based (sometimes it is called the memory-based methods), and whether this method can be applied in combination with the deep learning techniques becomes an interesting research problem. To answer the question, Travis Ebesu et al. proposed the Collaborative Memory Network (CMN) [14]. In the article of CMN, the

authors pointed out that the neighborhood-based methods such as KNN explicitly construct the user group with the common interest and then propagate the interests to those users groups that have close relationships. This method does not perform well in recommendation with varieties due to its limited focus on merely the k neighbors, and as a result, it can only capture the localized information. However, the methods that are based on the latent factors represent the user and item information at the low-dimension space, they do not account for the closely related user groups (e.g., the Top-k neighbors). CMN is designed to take the advantages of both the neighborhood-based method and the latent factor-based idea. It uses the memory network to unify the collaborative filtering methods into the same framework. To be more precise, the latent factor-based module in CMN is similar to a conventional matrix factorization, that is, a user matrix $\boldsymbol{U} \in \mathbb{R}^{(M \times d)}$ and an item matrix $\boldsymbol{V} \in \mathbb{R}^{(N \times d)}$ are formed. The novelty of CMN is that it has an additional user matrix, $\boldsymbol{C} \in \mathbb{R}^{(M \times d)}$, to represent the relationship between the neighbor users and the target users. Assuming $N(i)$ is the user set that has interacted with the item i , the following formula is used to calculate the interest similarities between the target user u and the neighbors $v \in N(i)$ on the item i .

$$q_{uiv} = (\boldsymbol{U}_u + \boldsymbol{V}_i)^\top \boldsymbol{U}_v. \tag{4.15}$$

Next, the matching scores for the neighbor users are transformed by using Softmax to normalize the weights for the neighbor users.

$$p_{uiv} = \frac{\exp(q_{uiv})}{\sum_{k \in N(i)} \exp(q_{uik})}, \ \forall v \in N(i). \tag{4.16}$$

After the weights are obtained, the neighborhood-based user vector can be formed by using the weights.

$$\boldsymbol{o}_{ui} = \sum_{v \in N(i)} p_{uiv} \boldsymbol{C}_v. \tag{4.17}$$

The predicted scores for the user u on the item i are generated from a neural network layer with the input of the two vectors being concatenated together.

$$\hat{y}_{ui} = \boldsymbol{w}_3^\top \phi(\boldsymbol{W}_1(\boldsymbol{U}_u \odot \boldsymbol{V}_i) + \boldsymbol{W}_2 \boldsymbol{o}_{ui} + \boldsymbol{b}). \tag{4.18}$$

In the above equation, $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d\times d}, \boldsymbol{w}_3, \boldsymbol{b} \in \mathbb{R}^d$ are the parameters of the neural network. ϕ is the activation function, and usually ReLU performs the best. The neighborhood searching module in CMN can be conveniently extended to "multiple hops" such that a more completed neighborhood-based user vectors can be obtained. With the 0-th hop being $\boldsymbol{z}_{ui}^0 = \boldsymbol{U}_u + \boldsymbol{V}_i$, the $k+1$-th hop can be formed by using the *k*-th hop and the output vector.

$$\boldsymbol{z}_{ui}^k = \phi(\boldsymbol{W}^k \boldsymbol{z}_{ui}^{k-1} + \boldsymbol{o}_{ui}^k + \boldsymbol{b}^k). \tag{4.19}$$

The matching score between the target user and the neighbor user is then replaced by $\boldsymbol{q}_{uiv}^{k+1} = (\boldsymbol{z}_{ui}^k)^\top \boldsymbol{U}_v$. Travis Ebesu et al. proved that increasing the hops can improve the model precision.

4.2 Deep Learning and Feature Interaction

The latent factor-based collaborative filtering method maps the user and item representations onto the low-dimension space. Its computation is straightforward so it is suitable to the recall models or the pre-ranking models. In the ranking models, to precisely delineate the users' preferences on items, contextual information is taken into account. For example, time, location, etc. Sometimes, even finer-grained feature interactions such as the one that is between user profile and item properties are leveraged. In the past, the feature interactions are designed manually or distilled automatically by using the gradient boosting decision tree. However, such methods are not generalizable into the feature combinations that have not appeared in the training set. Along with the proliferation of the deep learning technologies, the automatic feature interactions start to embrace new opportunities.

4.2.1 AFM Algorithm

Factorization machine (FM) is an algorithm that considers the second-order feature interactions in the model. However, when there are large volume of feature combinations that are not necessary in the training set, noise may be introduced, which can deteriorate the model performance. Jun Xiao et al. [74] proposed the Attentional Factorization Machine (AFM) to resolve the problem. Particularly, the second-order feature interaction module, $f_{\mathrm{PI}}(\mathcal{E})$, does not sum up all the feature combinations between one and another. Instead, the weighted sum is used.

$$f_{\text{Att}}(f_{\text{PI}}(\mathcal{E})) = \sum_{(i,j)\in\mathcal{R}_x} \alpha_{ij}(\boldsymbol{v}_i \cdot \boldsymbol{v}_j)\boldsymbol{x}_i\boldsymbol{x}_j. \tag{4.20}$$

In the above equation, α_{ij} is the scalar that represents the importance of the feature on (i, j), and it is generated from the attention network.

$$\alpha'_{ij} = \boldsymbol{h}^\top \text{ReLU}(\boldsymbol{W}(\boldsymbol{v}_i \cdot \boldsymbol{v}_j)\boldsymbol{x}_i\boldsymbol{x}_j + \boldsymbol{b}) \tag{4.21}$$

$$\alpha_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{(t,c)\in\mathcal{R}_x} \exp(\alpha'_{tc})}. \tag{4.22}$$

$\boldsymbol{W} \in \mathbb{R}^{k\times d}; \boldsymbol{b} \in \mathbb{R}^k; \boldsymbol{h} \in \mathbb{R}^k$ are the attention model parameters. The output of the second- order feature interaction module in the AFM model, $f_{\text{Att}}(f_{\text{PI}}(\mathcal{E}))$, is a *d*-dimension vector. It represents the second-order feature interactions after a weighted compression. The final prediction output of the AFM model is shown below.

$$\hat{y}_{\text{AFM}}(\boldsymbol{x}) = \boldsymbol{w}_0 + \sum_{i=1}^{n} \boldsymbol{w}_i\boldsymbol{x}_i + \boldsymbol{p}^\top \sum_{(i,j)\in\mathcal{R}_x} \alpha_{ij}(\boldsymbol{v}_i \cdot \boldsymbol{v}_j)\boldsymbol{x}_i\boldsymbol{x}_j. \tag{4.23}$$

The entire model structure of AFM is shown in Fig. 4.6.

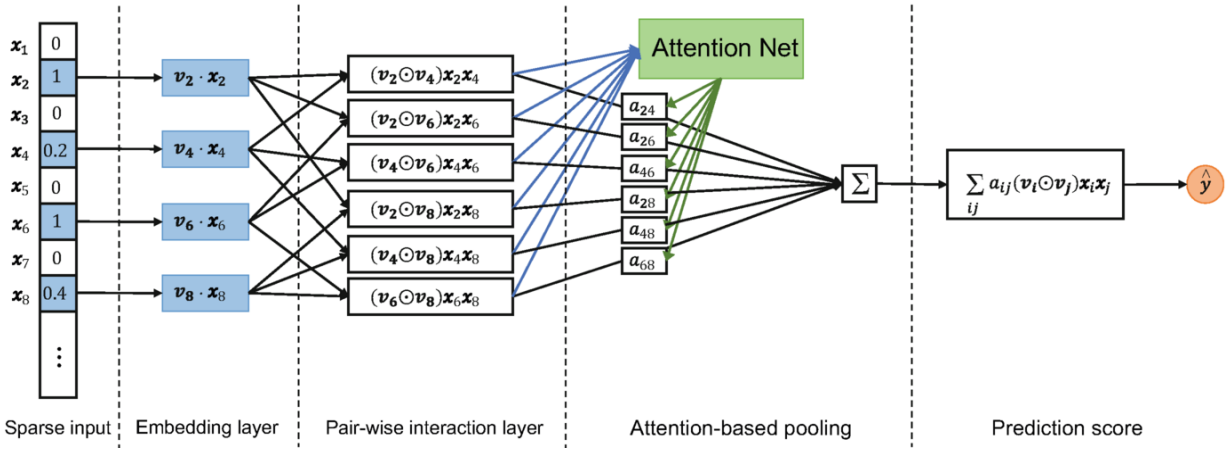


Fig. 4.6 AFM model structure

4.2.2 PNN Algorithm

The input samples in the recommender model that are used for ranking and click-through rate prediction are always sparse. For example, the user ID, item ID, discrete timestamps, categorical properties, etc., can all be used as

features in the sample data. And these highly sparse features can be categorized into different fields, and for each field, the one-hot code or multi-hot code is used to encode the original representation. The benefit of doing so is that although the number of features in each sample is changeable, the number of the feature field is fixed. Therefore, it is convenient to concatenate the latent factors corresponding to the feature field and use them as the input to MLP for the following computation. An illustrative example is shown below.

$$\underbrace{[0,1,0,0,0,0,0]}_{\text{Date=Week}} \quad \underbrace{[0,1]}_{\text{Gender=Male}} \quad \underbrace{[0,0,1,0,\cdots,0,0]}_{\text{Location=London}} \tag{4.24}$$

There are three different fields in the above example, that is, date, gender, and location. Each of the fields is represented by one-hot encoding. One of the straightforward approaches is to use the embedding lookup to find the low-dimension vectors for each feature field and then concatenate all of the vectors to feed into MLP for high-order feature interaction computation [81]. To use a more effective feature interaction, Yanru Qu et al. proposed Product-based Neural Networks (PNNs) [46], which innovatively added the explicit second-order feature interaction layer between feature fields. It is placed between the embedding layer and the MLP layer. The model architecture is shown in Fig. 4.7.

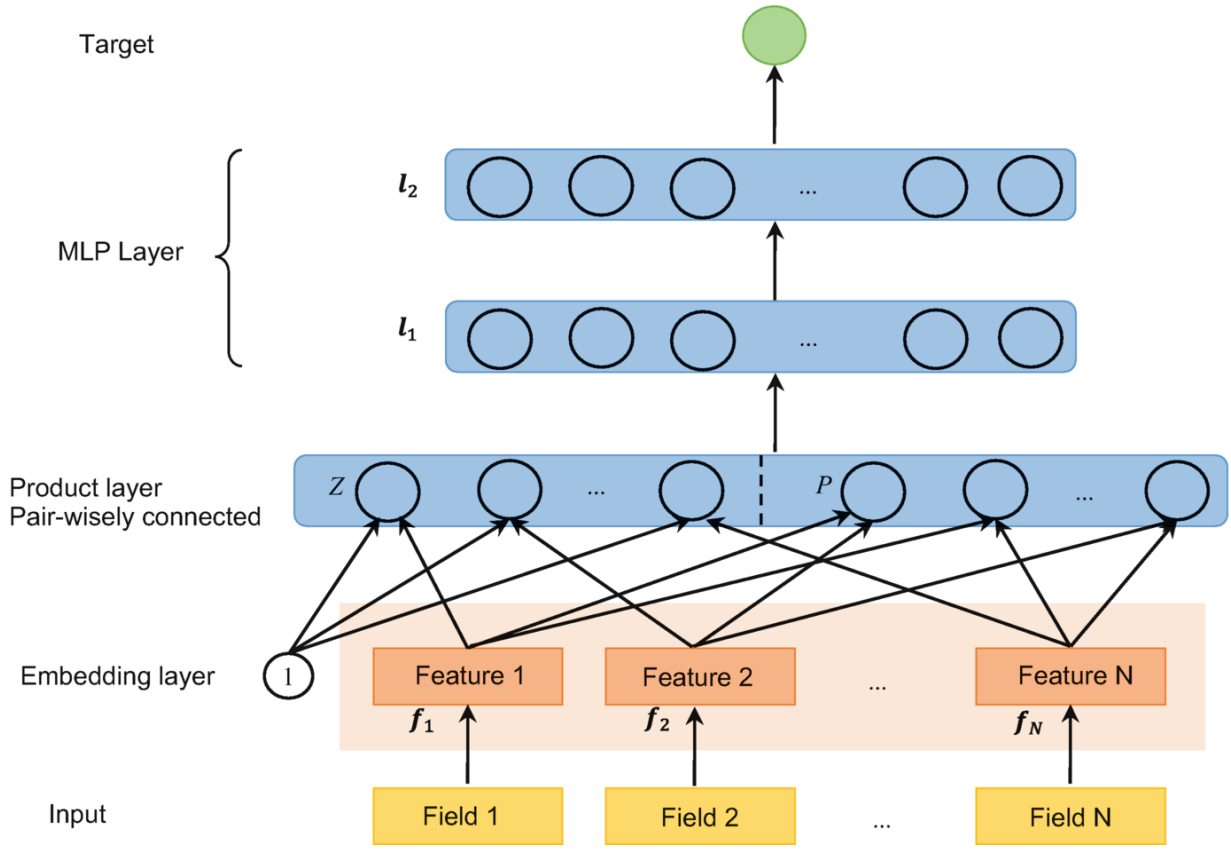


Fig. 4.7 PNN model architecture

The discussion here is mainly focused on the second-order feature interactions. From Fig. 4.7, it can be seen that the second-order feature interaction layer is composed of two parts, indicated by Z and P, respectively. Z is the interaction between each feature field and an all-one vector, and P is the interaction between every two feature fields. The outputs of Z and P are transformed via a linear layer, which are then mapped onto the latent factors that have fixed length.

$$\boldsymbol{l}_z = (\boldsymbol{l}_z^1, \boldsymbol{l}_z^2, \cdots, \boldsymbol{l}_z^k, \cdots, \boldsymbol{l}_z^{D_1}), \boldsymbol{l}_z^k = \boldsymbol{W}_z^k \odot \boldsymbol{Z} \tag{4.25}$$

$$\boldsymbol{l}_p = (\boldsymbol{l}_p^1, \boldsymbol{l}_p^2, \cdots, \boldsymbol{l}_p^k, \cdots, \boldsymbol{l}_p^{D_1}), \boldsymbol{l}_p^k = \boldsymbol{W}_p^k \odot \boldsymbol{P}. \tag{4.26}$$

The operation of the first layer in the MLP is

$$\boldsymbol{l}_1 = \mathrm{ReLU}(\boldsymbol{l}_z + \boldsymbol{l}_p + \boldsymbol{b}_1). \tag{4.27}$$

In the above equation, $\boldsymbol{b}_1$ is the vector that shifts D_1 dimensions. It can be intuitively understood that $\boldsymbol{l}_z$ and $\boldsymbol{l}_p$ are used to give Z and P a fully connected layer with D_1 units. To obtain Z and P, Yanru Qu et al. defined

two different types of interactions, inner product-based and outer product-based.

Inner Product-Based Interaction

In this situation, the content in Z is equivalent to concatenating vectors of all the feature fields. Therefore, the complexity of the parameters used in the $\boldsymbol{l}_z$-th layer is $O(D_1 NM)$, where *M* is the dimension of the feature embeddings. The content of P is the inner product of N fields, which are N^2 scalar values. Therefore, the complexity of the parameters at the $\boldsymbol{l}_p$-th layer is $O(D_1 N^2)$. If the parameter size is too large, the parameter matrix $\boldsymbol{W}_z^k$ can be assumed to be low rank, and it can be approximated as $\boldsymbol{W}_z^k = \boldsymbol{\theta}^k \boldsymbol{\theta}^{k\top}, \boldsymbol{\theta}^k \in \mathbb{R}^N$, such that the size of the parameters is not at the square level of N.

Outer Product-Based Interaction

The inner product of two vectors is a scalar, while the outer product is a matrix. That is

$$\boldsymbol{p}_{ij} = g(\boldsymbol{f}_i, \boldsymbol{f}_j) = \boldsymbol{f}_i \boldsymbol{f}_j^\top \in \mathbb{R}^{M \times M}. \tag{4.28}$$

The content of P is the outer product of the N fields. Therefore, the parameter size at $\boldsymbol{l}_1$-th layer is $O(D_1 N^2 M^2)$. This is apparently very big. To reduce the computational efforts and the size of the parameters, Yanru Qu proposed pooling the sum of the feature fields onto a vector, $\boldsymbol{f}_\Sigma = \sum_{i=1}^N \boldsymbol{f}_i$, where $\boldsymbol{f}_\Sigma \in \mathbb{R}^M$. Based on the vector, another outer product is performed, $\boldsymbol{p} = \boldsymbol{f}_\Sigma (\boldsymbol{f}_\Sigma)^\top$. By doing this, the parameter size of $\boldsymbol{l}_p$ becomes $O(D_1 M^2)$, and the total parameter size becomes $O(D_1 M^2 + D_1 NM)$.

4.2.3 Wide and Deep Algorithm

Wide and deep algorithm [10] was proposed by Google in the year of 2016. It is a recommendation algorithm that combines the deep learning technique. Since its birth, it has received compliment, and now it has become one of the main stream recommendation algorithms used in the industry. The wide and deep model emphasizes that a good recommender model should perform well in both memorization and generalization. Memorization refers to the model capability that it can capture the frequent common features in the data

and build the relationship between these features against the prediction label. This memorization part can be implemented by using a logistic regression model to learn the coefficients for the cross-product features. For example, if in the training set, users frequently click a news topic that is related to celebrity, then the cross-product feature of <user ID, topic ID, celebrity ID> is a good cross-product feature. Generalization is the capability of the model to transfer the existing patterns that have been learnt from the training data to those unobserved data (or those that do not frequently occur in the dataset). This can be implemented by using the latent factor embeddings plus the neural network. For example, if the users like the music from Jay Chou, it is very likely that the users are born in the 80s or 90s, and as a result, the users may also like the music from May Day.

Both memorization and generalization are important to recommender systems. If a model merely has the capability of memorization, the recommender system may only recommend items that are correlated to the users' historic interactions. As a consequence, the diversity of the recommendations is poor, and the holistic system is impacted negatively due to the Matthew effect. On the other hand, if a model is only capable of generalizing, due to the long-tail effect in the latent factors, these factors cannot be learnt effectively by the model, and thus the non-zero predictions can be still generated because of the existence of these factors. The consequence of such behavior is that the recommendation results may consist of many items that users do not have interest in. Wide and deep algorithm trains the linear model and the neural network model combinationally for having both the capabilities of memorization and generalization at the same time. Such idea is applied for the ranking stage in the recommender system of Google Apps.

The model structure is shown in the diagram of Fig. 4.8. The model is composed of two parts. The left side is the wide part, which is for memorization. It is generally a linear regression model, i.e., $y = \boldsymbol{w}_{\text{wide}}^{\top}\boldsymbol{x} + b$. The inputs are the original features and some cross-product features. $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_d]$. The right-hand side is the deep part, and it is for generalization. The input is sparse features, and the embeddings are found from the lookup table to obtain the low-dimension dense vectors $\boldsymbol{a}^{(0)}$ (in Google's real-world applications, $\boldsymbol{a}^{(0)}$ not only consists of the latent factors from the sparse features, but also the original dense features such as age, activities, etc.), which are then fed into the deep neural network (the

MLP layer is also termed as DNN module hereafter) for learning on the cross-product features $\boldsymbol{a}^{(l+1)} = f(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)})$. Here, l is the number of layers in the DNN, and f is the activation function. The inputs for the wide part and deep part are combined together for model building.

$$y = \sigma\left(\boldsymbol{w}_{\text{wide}}^{\top}\boldsymbol{x} + \boldsymbol{w}_{\text{deep}}^{\top}\boldsymbol{a}^{l_f} + b\right). \tag{4.29}$$

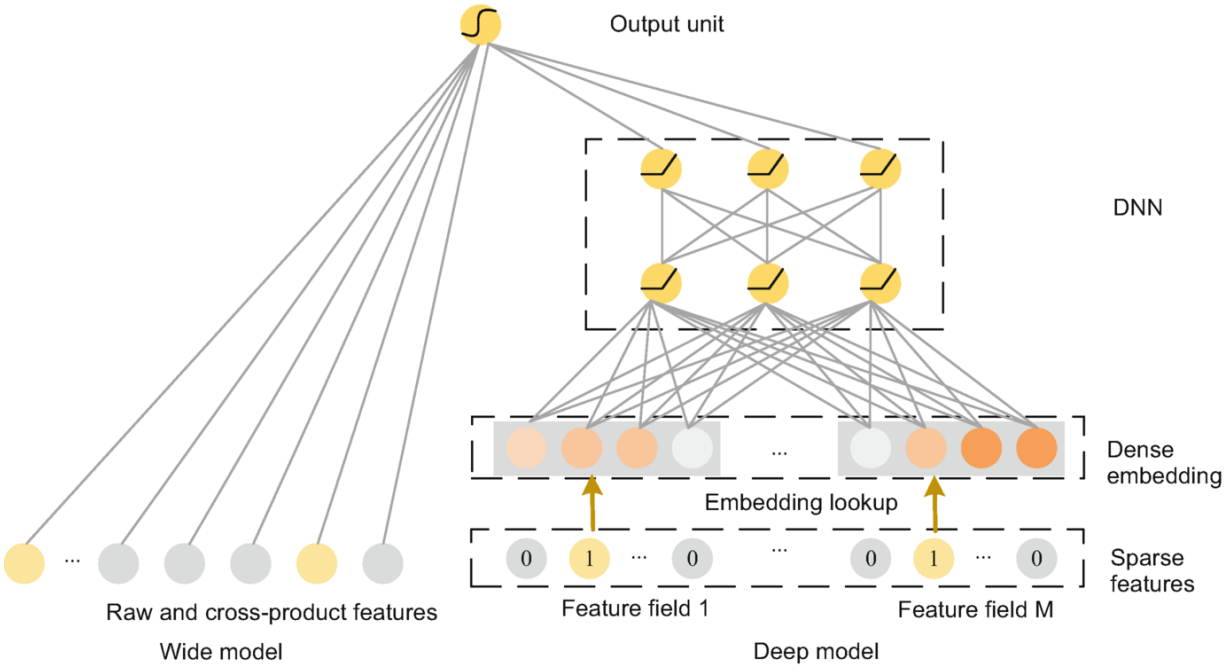


Fig. 4.8 Wide and deep model structure

In the above equation, $\boldsymbol{a}^{l_f}$ is the value after the activation function of the deep part. σ is the Sigmoid activation function. When doing the joint training, for the wide part, FTRL is recommended [41] to be used as the optimizer, while for the deep part, AdaGrad [13] is recommended.

4.2.4 DeepFM Algorithm

In the deep part of the wide and deep model, DNN can learn the high-order feature interactions automatically. However, it cannot guarantee that the useful low-order feature interactions are well learnt. In the meantime, the wide part relies on human beings to extract the cross-product features for the low-order feature interactions, but this manual method does not scale for all the possible feature interactions. The classic factorization machine is able to explicitly model the second-order feature interactions and capture all the possible ones. To mitigate the issue that the existing models tend to learn

either the high-order feature interactions, the linear features, or rely on the manually extracted low-order feature interactions, Huifeng Guo et al. [18] proposed the DeepFM algorithm, which combines factorization machine and MLP into the same model structure so that it is capable of modeling the low-order feature interactions (from the FM part) and the high-order feature interactions (from the DNN part). The model structure of DeepFM is shown in Fig. 4.9. Similar to wide and deep, the final output of DeepFM is the combination of the two parts.

$$\hat{y} = \text{Sigmoid}(\hat{y}_{\text{FM}} + \hat{y}_{\text{DNN}}). \tag{4.30}$$

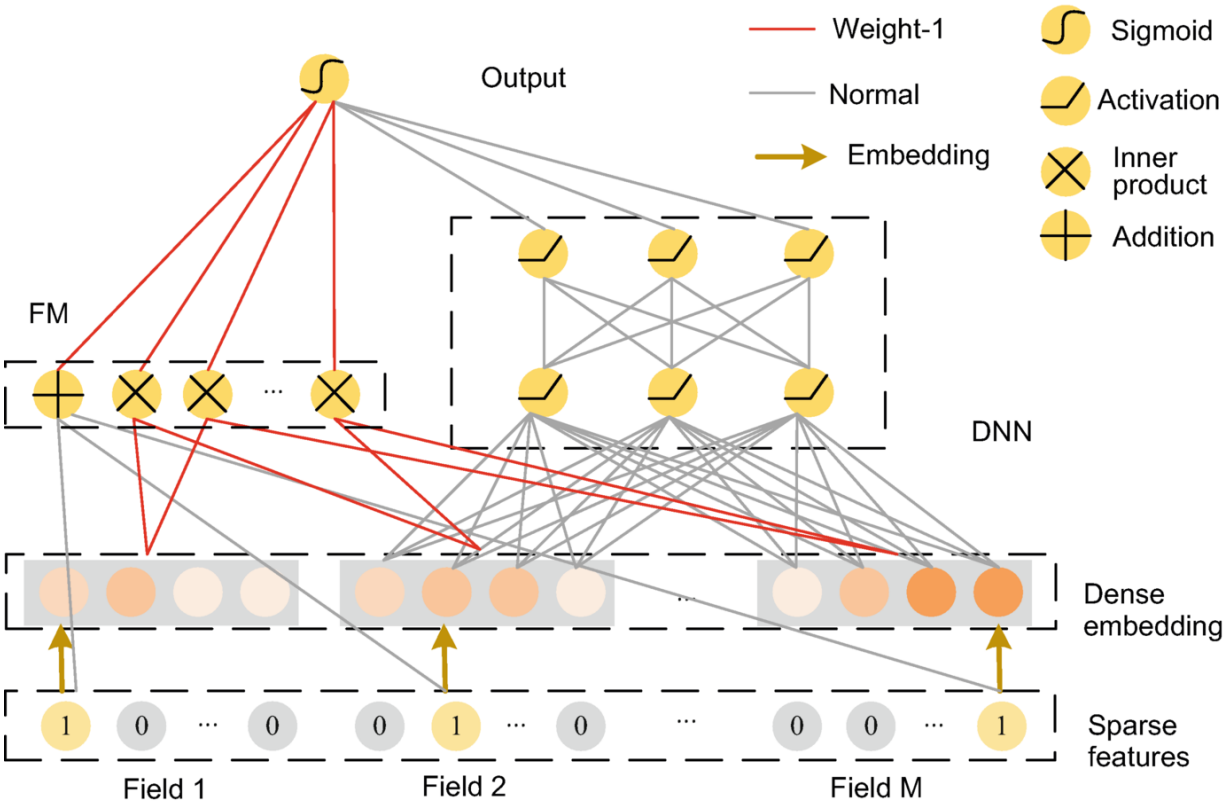


Fig. 4.9 DeepFM model structure

Interestingly, Huifeng Guo et al. experimentally demonstrated that by sharing the latent factors for both the FM part and the DNN part is more effective than using two separate sets of latent factors. That is to say, the high-dimension sparse features can be obtained by taking the embedding lookup operation from the corresponding latent factors, and these latent factors are fed into the FM part to compute the second-order feature interactions and concatenated according to the feature sequence. In the DNN part, the high-order feature interactions are learnt. The two parts are trained

jointly in an end-to-end fashion. As a result, DeepFM does not need a FNN module [81] that uses the pre-trained FM vectors for the DNN initialization. This proves that sharing the latent factors for both low-order and high-order feature interactions is advantageous.

4.2.5 DCN Algorithm

It was realized by the researchers that although the linear model is simple, scalable, and explainable, it does not favor the expressibility need. Also, the linear model relies on the engineer-generated feature interactions, and it cannot be well generalized to the feature interactions that do not exist in the training set. Until now, the capability of learning high-order feature interactions in a model still requires DNN. Therefore, how to effectively extend DNN becomes an interesting research topic. Considering that DNN can merely implicitly model the high-order feature interactions and its training process and outcome is a black box, it cannot be guaranteed that the DNN model can learn the completed set of the high-order feature interactions. Is it possible to design a novel deep neural network that can learn the high-order feature interactions in a better way? Ruoxi Wang [64] proposed the DCN model that has the advantages as following: It can explicitly capture the high-order feature interactions and the order is controllable. The model structure is similar to wide and deep, DeepFM, etc., which has two parts. In this chapter, merely the cross network part of DCN is discussed. The structure of cross network is shown in Fig. 4.10. The cross network is designed to make the k-th layer in the cross network cover the $k+1$-th layer for the feature interactions.

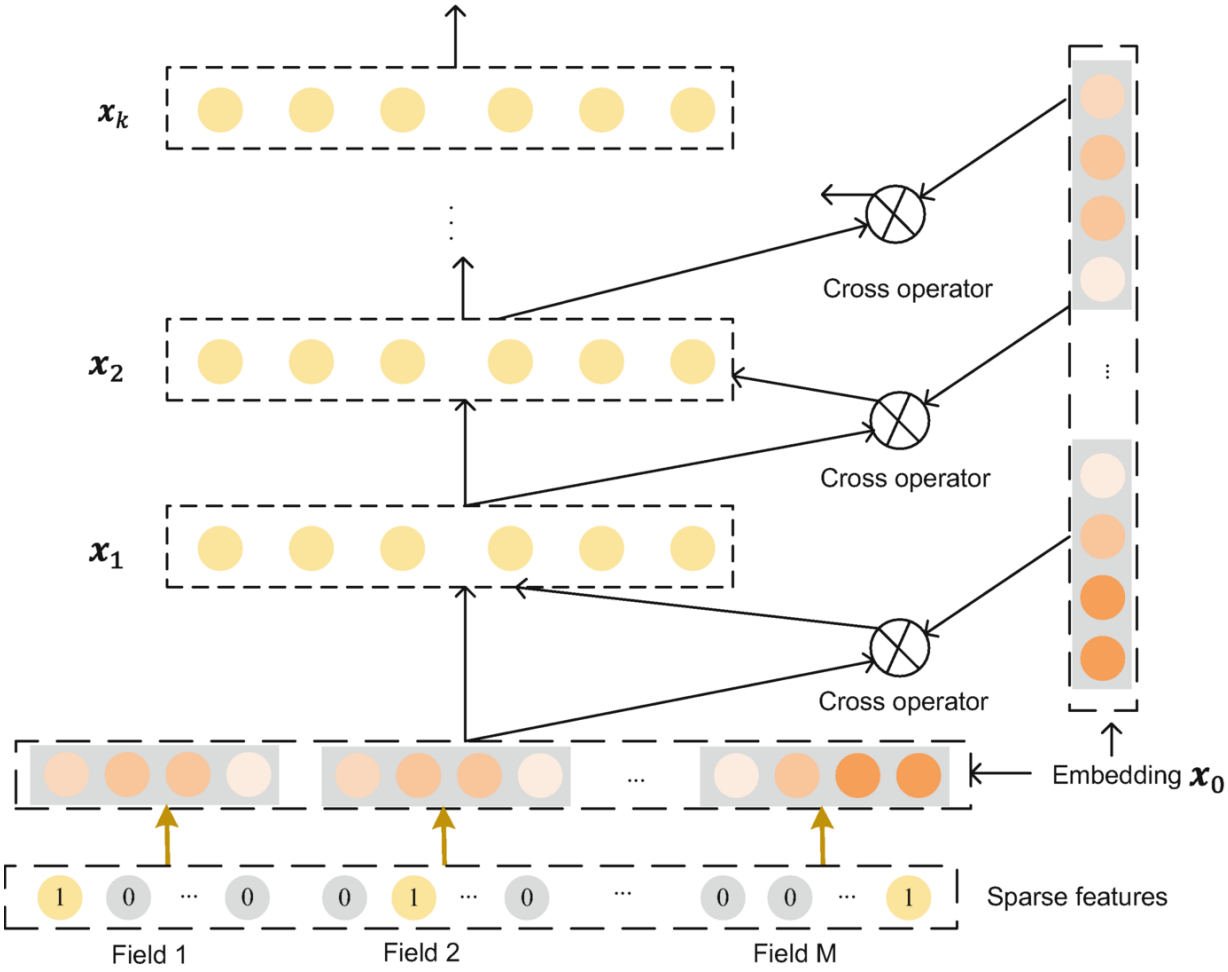


Fig. 4.10 Cross network in DCN

Every single layer is computed from the hidden layer states and the original input feature latent factors in the precedent layer via a cross network operator. Therefore, every time the cross network is appended with an additional layer, and the number of order for feature interactions is added by one. The cross network operator $f(\cdot)$ is defined as below.

$$\boldsymbol{x}_{l+1} = f(\boldsymbol{x}_l, \boldsymbol{w}_l, \boldsymbol{b}_l) + \boldsymbol{x}_l = \boldsymbol{x}_0 \boldsymbol{x}_l^\top \boldsymbol{w}_l + \boldsymbol{b}_l + \boldsymbol{x}_l. \tag{4.31}$$

In the above equation, $\boldsymbol{x}_0 \in \mathbb{R}^d$ is the latent factor after the concatenation. $\boldsymbol{x}_l$ is the hidden state in the l -th layer of the cross network. $\boldsymbol{w}_l, \boldsymbol{b}_l \in \mathbb{R}^d$ indicates the learnable parameters in the l -th layer. The benefits of the cross network are that it is computationally efficient, size of the parameters is small, and it is highly scalable. For example, there are only $2d$ parameters in each layer of the cross network. At the same time, the number of layers of the cross network strictly controls the order of feature

interactions. Ruoxi Wang et al. provided the mathematical proof that can be referenced in the publication of DCN [64]. Similar to wide and deep, DCN also combines the outputs from the cross network and the fully connected neural network as the final output.

$$\hat{y} = \text{Sigmoid}(\boldsymbol{w}^\top [\boldsymbol{x}_{\text{LC}}, \boldsymbol{x}_{\text{DNN}}]). \tag{4.32}$$

In the above equation, x_{LC} is the output features from the cross network, and x_{DNN} is the ones from the full-connected network.

4.2.6 xDeepFM Algorithm

The proposal of DCN inspired the researchers in the field. Jianxun Lian et al. [35] found that although the cross network in DCN is ideal for its simplicity and computational efficiency, it has the drawback that the final hidden state of the cross network is only applicable as a scaling form of the original latent factor $\boldsymbol{x}_0$. For the sake of simplicity, assuming the activation function in the cross network is an identity function, it can be derived that

$$\boldsymbol{x}_1 = \boldsymbol{x}_0(\boldsymbol{x}_0^\top \boldsymbol{w}_1) + \boldsymbol{x}_0 = \boldsymbol{x}_0(\boldsymbol{x}_0^\top \boldsymbol{w}_1 + \mathbf{1}) \triangleq \alpha_1 \boldsymbol{x}_0. \tag{4.33}$$

In the above equation, $\alpha_1 = (\boldsymbol{x}_0^\top \boldsymbol{w}_1 + \mathbf{1})$ is a scalar. It can be mathematically inducted that assuming that $\boldsymbol{x}_k = \alpha_k \boldsymbol{x}_0$ holds for all the possible $k \leq i$, when $k = i + 1$, we have

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_0(\boldsymbol{x}_i^\top \boldsymbol{w}_{i+1}) + \boldsymbol{x}_i = \boldsymbol{x}_0((\alpha_i \boldsymbol{x}_0)^\top \boldsymbol{w}_1 + \alpha_i) \triangleq \alpha_{i+1} \boldsymbol{x}_0. \tag{4.34}$$

Therefore, $\boldsymbol{x}_{i+1}$ is still scaling form of $\boldsymbol{x}_0$. It is worth mentioning that this does not mean that the output from the cross network has a linear relationship with the original latent factors because the corresponding scalar α_{i+1} is dynamically correlated to the data samples $\boldsymbol{x}_0$. And thus, the form of $\boldsymbol{x}_{i+1}$ is limited.

To explore a more flexible explicit high-order feature interactions, Jianxun Lian et al. [35] proposed the new neural network structure, i.e., the Compressed Interaction Network (CIN). CIN is ideated based on two findings. First, the bit-wise feature interactions can be replaced by the vector-wise ones. Given that the latent factors represent the feature field, the interactions among the feature field can be useful, while the interactions among the elements in a feature field may not be useful. Also, DNN uses the bit-wise full connection in the network. Theoretically, it can be used to model any complex function, but it is difficult to learn the model parameters

properly. Particular, on the recommendation related datasets where feature interactions are obvious, whether DNN can effectively model the high-order feature interactions or not is uncertain. Similar to the idea of CIN, Alex Beutel et al. [3] proposed the vector-wise interaction in the article about Latent Cross, and it was proved by a data-driven experiment that to train a DNN for feature interactions is not straightforward. Second, given that DNN can automatically extract complex features, e.g., the automatic feature extraction from the unstructured raw data of images, text, and audio, is it possible to treat the raw feature interactions as unstructured data and use DNN to automatically extract the useful interactions from them? The answer is possible, and this is where the term of "compression" in CIN comes from.

Specifically, the input to CIN is no longer vectors. It is matrix instead. $\boldsymbol{X}^0 \in \mathbb{R}^{M \times D}$, where M is the number of feature fields, D is the dimensionality of the embedding vectors, and the i-th row in $\boldsymbol{X}^0$ is the embedding of the i-th feature field, $\boldsymbol{X}^0_{i,*} = \boldsymbol{e}_i \in \mathbb{R}^D$. The k-th hidden layer in CIN is also a matrix, $\boldsymbol{X}^k \in \mathbb{R}^{H_k \times D}$, in which H_k is the number of the features in the k-th layer. The structure of $H_0 = M$ $\boldsymbol{X}^0$ and $\boldsymbol{X}^k$ is illustrated in Fig. 4.11. Based on the hidden layer matrix in the k-th layer and the original feature matrix, the intermediate output $\boldsymbol{Z}^{k+1}$ can be obtained. This step is not parameterized. $\boldsymbol{Z}^{k+1}$ is the original content after all the features are interacted, and it can be treated similarly to the raw data of an "image". As shown in Fig. 4.11, from the intermediate output $\boldsymbol{Z}^{k+1}$, there are H_{k+1} feature maps extracted, which can be used as the hidden layer matrix in the $k+1$-th layer.

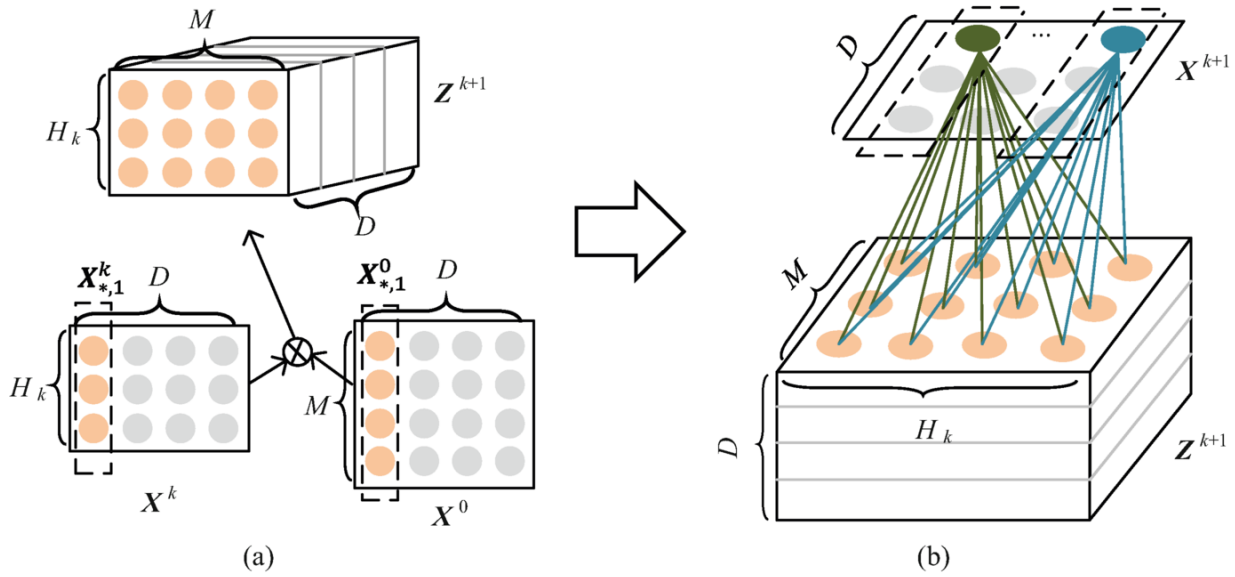


Fig. 4.11 The computational procedure of CIN

The operation inside the compressed interactions can be formalized as below.

$$\boldsymbol{X}_{h,*}^{k} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{M} \boldsymbol{W}_{ij}^{k,h} (\boldsymbol{X}_{i,*}^{k-1} \odot \boldsymbol{X}_{j,*}^{0}). \tag{4.35}$$

In the above equation, *odot* is the Hadamard product of the two vectors, i.e., $a_1, a_2, a_3 \odot b_1, b_2, b_3 = a_1 b_1, a_2 b_2, a_3 b_3$. $\boldsymbol{W}_{*,*}^{k,h} \in \mathbb{R}^{H_{k-1} \times M}$ is the parameters for the *h*-th feature map in the *k*-th layer. The computation can be illustrated in the diagram in Fig. 4.11.

Figure 4.12 demonstrates the overall architecture of CIN. Due to the interaction between the input feature embedding matrix for each layer and the one in the previous layer, the order of feature interactions increments when the network is appended with an additional layer. To make sure that feature interactions from low order to high order are all sufficiently captured, the feature map at each layer is through a sum pooling before it is used in the prediction module at the last stage. It is worth noting that CIN has similarities with both RNN and CNN. The commonality between CIN and RNN is that the computational output of each hidden layer depends on the activation value of the last layer and an additional input. The two are different in the aspect that the inputs for RNN at each time are new (e.g., the words in a sentence) so that the parameters for each neural network unit are shared, while the inputs for CIN are fixed (they are always the feature

embeddings) so that the parameters for each neural network unit are fresh. The common functionality between CIN and CNN is that the intermediate output in CIN $\boldsymbol{Z}^{k+1}$ (shown in Fig. 4.11) can be treated as an image, from which H_{k+1} feature maps are learnt. Each feature map is associated with a convolutional kernel with parameter size $H_k \times M$ and dimensionality D.

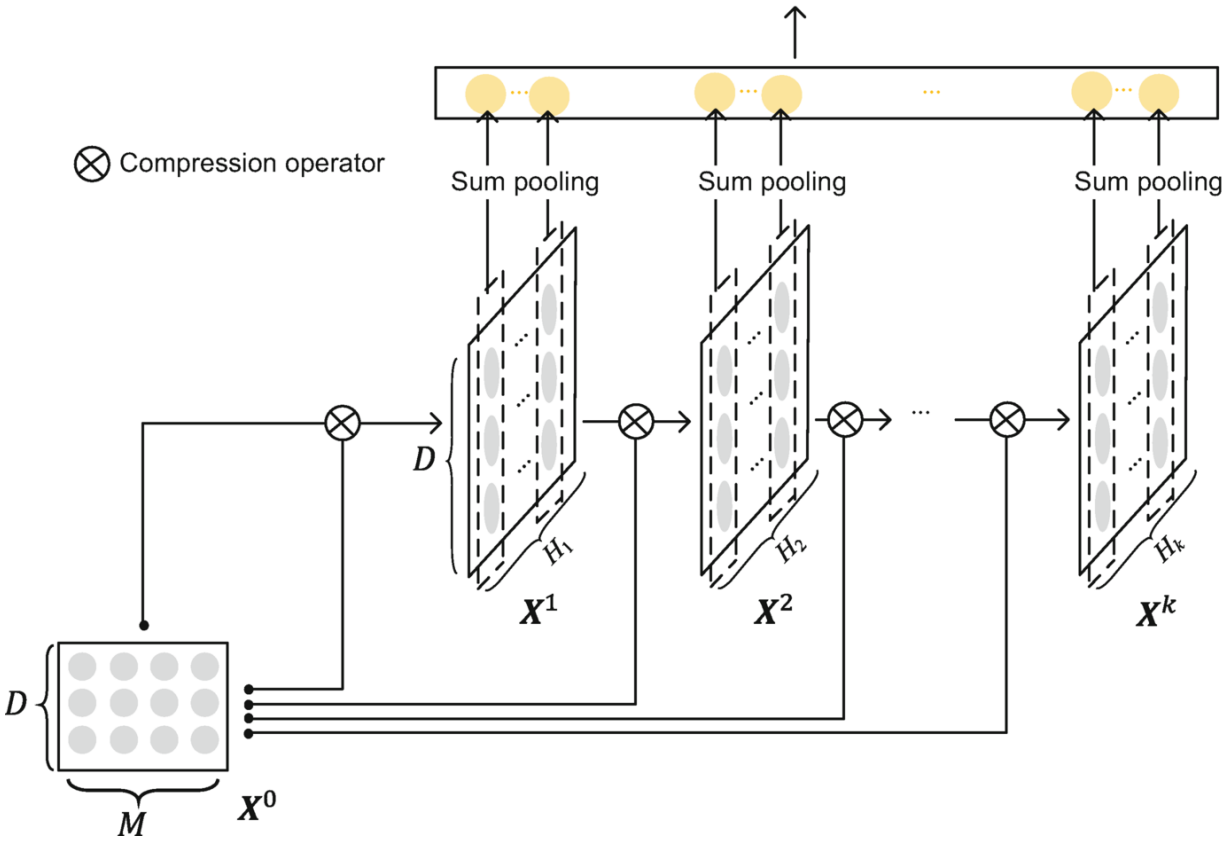


Fig. 4.12 Overall architecture of CIN

In the last, similar to wide and deep, Jianxun Lian et al. [35] combine the output from the linear part, DIN, and DNN, to feed into the prediction part.

$$\hat{y} = \text{Sigmoid}(\boldsymbol{w}^\top [\boldsymbol{a}, \boldsymbol{x}_{\text{DNN}}, \boldsymbol{x}_{\text{CIN}}]). \tag{4.36}$$

In the above equation, $\boldsymbol{a}$ is the raw feature, $\boldsymbol{x}_{\text{DNN}}$ is the last hidden layer in DNN, and $\boldsymbol{x}_{\text{CIN}}$ is the vector generated by sum pooling the output from CIN. The algorithm is called eXtreme Deep Factorization Machine (xDeepFM).

4.2.7 AutoInt Algorithm

Along with the success of the transformer-based model in natural language processing, the application of transformer techniques on feature interactions

has been researched by scholars. Weiping Song et al. proposed using the core component, Multi-head self-attention (MSA) in the transformer model into the feature embeddings of a recommender model, to automatically learn the high-order feature interactions. The embedding representation of a feature sample is $\boldsymbol{e}_x = [\boldsymbol{e}_1; \boldsymbol{e}_2; \cdots; \boldsymbol{e}_M]$, where $\boldsymbol{e}_m = \boldsymbol{v}_m \boldsymbol{x}_m$. There are two important units in MSR—the self-attention part and the multi-head part. The self-attention module is aimed at improving the latent factor representation for the features so that the new vectors can properly contain the information from the other features (this is called the contextual vector representation) instead of just the IDs. This is essentially the feature interaction process in the form of <query, key, value>. Each of the features uses its own vector representation as the query and computes the similarities with the vectors of the other features for key. The similarities are used as the weights for applying the feature vectors on the self-vectors as the value. Using the m-th feature as an example, the similarity between it and the *k*-th feature can be computed as below.

$$\alpha_{m,k}^{(h)} = \frac{\exp(\psi^{(h)}(\boldsymbol{e}_m, \boldsymbol{e}_k))}{\sum_{l=1}^{M} \exp(\psi^{(h)}(\boldsymbol{e}_m, \boldsymbol{e}_l))} \tag{4.37}$$

$$\psi^{(h)}(\boldsymbol{e}_m, \boldsymbol{e}_k) = \langle \boldsymbol{W}_{\text{Query}}^{(h)} \boldsymbol{e}_m, \boldsymbol{W}_{\text{Key}}^{(h)} \boldsymbol{e}_k \rangle. \tag{4.38}$$

In the above equation, $< \cdot >$ is the inner product (it can be combined with a scaling factor). $\boldsymbol{W}_{\text{Query}}^{(h)}$, $\boldsymbol{W}_{\text{Key}}^{(h)} \in \mathbb{R}^{d' \times d}$ is the transformation matrix. After that, the feature vector m is updated.

$$\overline{\boldsymbol{e}}_m^{(h)} = \sum_{k=1}^{M} \alpha_{m,k}^{(h)} (\boldsymbol{W}_{\text{Value}}^{(h)} \boldsymbol{e}_k). \tag{4.39}$$

Since the vector $\overline{\boldsymbol{e}}_m^{(h)} \in \mathbb{R}^{d'}$ is a combination of feature *m* and the other features, it can be regarded as one round of feature interaction. In addition, the multi-head mapping is used to indicate that such interaction is executed for multiple times with different sets of parameters, and each time it is operated a head is used to learn the features from a different perspective. The overall vector representation of feature *m* is the concatenation of the output from *H* attention heads.

$$\overline{\boldsymbol{e}}_m = \overline{\boldsymbol{e}}_m^{(1)} \oplus \overline{\boldsymbol{e}}_m^{(2)} \oplus \cdots \oplus \overline{\boldsymbol{e}}_m^{(H)}. \tag{4.40}$$

To well maintain the original latent factors for *m*, the final output is processed by using a residual network, to fuse the new vector $\overline{e}_m$ and the original one $\boldsymbol{e}_m$.

$$\boldsymbol{e}_m^{\text{Res}} = \text{ReLU}(\bar{e}_m + \boldsymbol{W}_{\text{Res}}\boldsymbol{e}_m). \tag{4.41}$$

The process of MSA can be repeated for multiple times, to get the high-order feature interaction information. Due to the fact in each step of the MSA transformation the features are combined with the information from other features, the feature vector can be concatenated in the last stage to get the prediction value via a logistic regression module.

$$\hat{y} = \sigma(\boldsymbol{w}^\top(\boldsymbol{e}_1^{\text{Res}} \oplus \boldsymbol{e}_2^{\text{Res}} \oplus \cdots \oplus \boldsymbol{e}_M^{\text{Res}}) + b). \tag{4.42}$$

4.2.8 Additional Thoughts on Feature Interaction

Effectively learning feature interactions for ranking in recommender systems, CTR prediction, etc., is a big challenge. There is a plenty of research work in this topic. Due to the limit of space, they are not reviewed here. Readers can check the references for further readings. For example, the research team in Sina AI Lab proposed the FiBiNet algorithm [26] that enhances the existing feature interaction methods in two folds. First, it replaces the conventional inner product and Hadamard product by bilinear product, to capture the fine-grained feature interactions. SENET module is introduced to dynamically adjust the weights of the feature vectors. Noah Ark Lab at Huawei proposed the AutoFIS algorithm [38], which is capable of capturing the meaningful feature interactions while abandoning the meaningless or even the noisy ones, and it got improvement in terms of both model performance and efficiency over the existing methods. Alimama proposed the DIN framework where the CAN model was introduced [4]. It formalized the feature interactions into feature transformation in DNN. For example, to get the latent factor of feature $\boldsymbol{A}$ against the target feature $\boldsymbol{B}$, a set of parameters can be used correspondingly in a DNN. When the original latent factor of $\boldsymbol{A}$ is used as input to the DNN, the output is the interaction between $\boldsymbol{A}$ and $\boldsymbol{B}$.

4.3 Graph Representation Learning and Recommender System

Algorithms of RNN, CNN, etc., assume the input data of the model to be in a space where the input vector positions are ordered. For example, the input image that is the input of CNN has the pixel values that are positioned in an Euclidean space, that is, the pixels are placed among the others with the relative positions of its surroundings. Similarly, the words in a sentence that are processed by RNN are also ordered in sequences. However, there is a commonly seen data model, the graph representation, where there is no such ordered space. Examples of such unordered data model include social media network, knowledge graph, chemical molecules, DNA structure, etc. The nodes on a graph can have different neighbors, and the neighbors do not have strong spatial correlations among each other. In most situations, a spatial exchange of neighbors in a graph does not change the physical meaning of the graph. Graph mining is a challenging task. However, given that a graph reflects the raw and unprocessed relationship between data points, by designing efficient models to process complex data, significant progress can be made in many application domains.

Many data can be represented in graph for recommender system applications. The user–item interaction data can be described as a bipartite graph. The classic neighborhood-based collaborative filtering method models the user-to-user similarities based on the neighborhood similarities on such bipartite graph. The co-occurrence of items can be constructed as an item-to-item graph. For example, edge in such graph indicates that the two items are purchased or clicked on by users frequently. The items can also be connected as an item-based knowledge graph to thoroughly delineate the item-to-item and item-to-property relationships. In addition, the user-to-user relationship can also be connected by leveraging the social connections because acquaintances may have similar interests, and the influencing nodes in the social media network may impact on the interests of other users.

By nature of the powerful representative learning capability of deep learning, applying deep learning in the graph data model attracts attentions from researchers. In this section, the application of graph neural network in recommender system is reviewed. To better present the content related to this topic, the following subsections will start with the introduction of the graph-based embeddings in the early days and then discuss the applications

of graph neural network in the recommender systems, to help the readers get a completed and systematic understanding.

4.3.1 Graph Embedding and Fundamentals of Graph Neural Network

Graph Embedding

A typical graph can be a complicated data structure. How to efficiently represent the node information of a graph is a challenging task. Inspired by the Word2Vec algorithm [42], it was found that using the low-dimension vector to represent the graph nodes is an effective method. Essentially, this is a self-supervised learning method based on the graph data structure. For a given graph, the algorithm learns a low-dimension embedding for each node ID in the graph so that the proximity of the neighboring notes can be obtained by using the inner product or the Cosine similarities of the embeddings. In this task, the most critical point is how to design the supervisory signal for the proximity of the neighboring nodes. In the subsection, four representative algorithms for resolving the issues, namely, DeepWalk, Node2Vec, LINE, and SDNE, are reviewed. Among the four algorithms, DeepWalk and Node2Vec make use of the random walk technique to generate the supervisory signal, and LINE and SDNE regress the first-order and second-order node proximity directly.

DeepWalk [44] is a two-stage model. Its idea is fairly easy to understand. In the first stage, starting from each node, the random walk algorithm is performed on the graph, to obtain the path $p = <v_1, v_2, \cdots, v_n$ with the length of *n*. In the second stage, similar to the Word2Vec model used in NLP, each path is treated as a sentence and nodes in the path are treated as words. For the nodes that co-occur in any limited window, the Skip-gram method is applied to get the latent factors. Usually, Word2Vec module in the library of Gensim is used to implement the second stage.

In NLP, the words in the same sentence have close semantic relationship. However, in the DeepWalk algorithm, given that the paths are obtained via random walk, the nodes in a path may not reflect the graph structure precisely. Aditya et al. then proposed Node2Vec [17] to resolve the issue. Considering that there are two types of searches for graph, i.e., breadth-first search (BFS) and depth-first search (DFS), DeepWalk can be regarded as a DFS-based approach, but this may not be the optimal one. A combinational approach that takes both BFS and DFS in generating the paths can help

retrieve the structural pattern in the graph. As shown in Fig. 4.13, assuming the last walk is from node t to node V, the current step is to hop from node V to the next one; instead of randomly selecting the next node as that in DeepWalk, Node2Vec categorizes the neighboring nodes of V into 3 different groups that are associated with different probability values. The first group is the nodes t, meaning that the hop steps back from V to the last one. The second group is the nodes $X1$, and this group of nodes are connected to both V and t. The third group of nodes are X_2 and X_3, which are the neighbors of V but not the ones of t. The probabilities corresponding to the three groups of nodes are $1/p$, 1, and $1/q$. After the traversal step on the graph, the second-stage operations in Node2Vec are the same as those in DeepWalk.

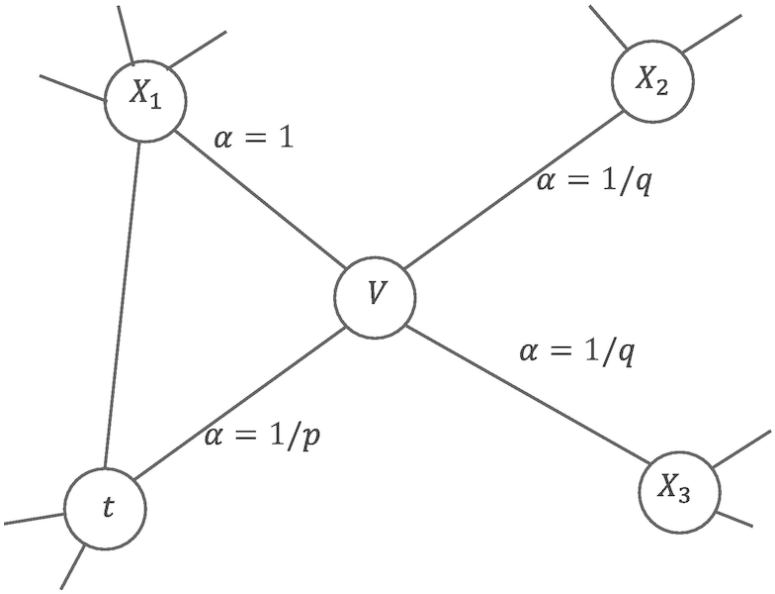


Fig. 4.13 The probability distribution of the random walk in Node2Vec

The idea in LINE [54] is different from DeepWalk and Node2Vec. It does not leverage the random walk technique to obtain the proximity of graph nodes for structurally modeling the graph. Instead, it uses the low-dimension representation to directly indicate the first-order and second-order node proximity. Here, the first-order proximity refers to the two nodes that are directly connected, and naturally, the vector representations of the two nodes are similar. The second-order proximity refers to the similarity between the neighbor nodes of two nodes under comparison. If the two nodes do not have direct connections, they may still have similar representations. Modeling the first-order proximity is rather straightforward.

It can be done simply by maximizing the inner product of the low-dimension representations $\boldsymbol{u}_i, \boldsymbol{u}_j$ (if the l1 norm of the vectors is 1, then the inner product is equivalent to the cosine similarity). Directly modeling the second-order proximity is hard because the neighbor node set of a node may have many members. An enumeration-based method may incur a significant amount of computational efforts. LINE algorithm nicely leverages the context vector to resolve the problem. That is, each of the nodes i has a self-vector $\boldsymbol{u}_i$ and a context vector $\boldsymbol{c}_i$, and then the second-order proximity can be converted into a problem that requires only enumerating the edges $< i, j >$ such that the inner product of the self-vector and the context vector of node i is maximized. Essentially, this operation is a propagation of node relationship. If two nodes have a large overlap of the neighbor set, it means that the two vectors of the nodes and their context vectors have high values of the inner product, and thus the self-vectors of the two nodes are similar to each other.

SDNE [57] shares a similar idea to LINE. Differently, SDNE uses neural network to learn the latent vector representations, while LINE, DeepWalk, and Node2Vec all use the lookup table for node ID to get the latent vector representation. SDNE leverages the structure used in AutoEncoder, where the neighbors of a node are taken as network input to a MLP for generating the low-dimension vector representation. The first-order proximity of the nodes can then be represented by these vectors. In the meantime, there is a decoder module in SDNE to reflect the neighbor relationship among the nodes by using the low-dimension vectors. The idea applied in this task is similar to the use of self-vectors and context vectors in LINE. It is worth mentioning that the idea of SDNE is already close to that of graph neural network.

There are many use case examples of the graph embedding methods. Jizhe Wang et al. [62] proposed their work, EGES, at KDD'18 about how the DeepWalk-based algorithm is applied for product embeddings in the Taobao front-page recommendations. In this method, the user behavioral sequences are first constructed as a product-to-produce directed graph. For users' repeated behavior in a session (a session is defined as the sequential behavior of a user within one hour), when enumerating the products, a directed edge can be added to construct a directed graph. The edges in the graph are weighted, and the weights are the co-occurrence of the two products in the historical behavior of the user. In this product graph, the

edge-based random walk that normalizes the weights as probabilities is performed, to obtain a sequence of paths. Then the algorithm of Skip-gram is performed to get the latent vectors of the products. The idea is in fact used in DeepWalk, but the disadvantage of DeepWalk is that it cannot be applied to the cold-start products. Considering the products' properties such as category, merchant, price, etc., fusing the property information into the embedding representation is beneficial to resolving the cold-start problem. Assuming that there are *n* properties for products v, the corresponding latent vector can be represented as $\boldsymbol{W}_v^s$ according to the product ID. To differentiate the importance of properties, there is another parameter vector $\boldsymbol{a}_v \in \mathbb{R}^{n+1}$ for weighting. The final vector representation of a product is the weighted sum of the latent vectors for all the properties.

$$\boldsymbol{H}_v = \frac{\sum_{i=0}^{n} \mathrm{e}^{\boldsymbol{a}_v^i} \boldsymbol{W}_v^i}{\sum_{j=0}^{n} \mathrm{e}^{\boldsymbol{a}_v^j}}. \tag{4.43}$$

Then the Skip-gram algorithm can be used to make sure that the vector $\boldsymbol{H}_v$ for a product in each path is similar as much as possible to the context vector $\boldsymbol{Z}_u$ of the other products *u* in the session. The latent vector $\boldsymbol{H}_v$ after the training process can be used in two ways in a recommender system. One is for the item-to-item-based product recalling, and another is for improving the precision in the ranking stage as product features.

Graph Neural Network

Thomas n. Kipf and Max Welling proposed GCN, and it generated much interest in researching the graph neural network-related topics. GCN uses the idea of CNN directly onto a graph such that the information of nodes and their neighbors can be fused. The convolution operations can recursively be appended to make each node be combined with the information of neighbors from a distance of multiple hops. The method that GCN uses to generate the latent vectors is shown below.

$$\boldsymbol{H}^{(l+1)} = \sigma\big(\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\big). \tag{4.44}$$

In the above equation, $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ is the normalized adjacent matrix. $\boldsymbol{A}$ is the adjacent Matrix, and each node has an edge that connects to itself. $\boldsymbol{D}$ is the diagonal matrix generated from $\boldsymbol{A}$ from each node. $\boldsymbol{W}^{(l)}$ is the parameter for the *l*-th layer neural network. $\boldsymbol{H}^l$ is the features of each

node. GCN has a severe scalability issue: Along with the increase of the layer number, the size of node neighbors increases exponentially. This makes it a computationally challenge for being practical. As a result, there are different methods proposed to enhance the scalability of GCN. In this subsection, three most representative methods are reviewed.

(1) GraphSage Algorithm [19]

William L. Hamilton et al. proposed a neighbor-based sampling method, GraphSage, to control the number of neighbors in each convolutional layer. This method was successfully used in the recommender system of Pinterest as PinSage and implemented on a distributed system. GraphSage redesigned the convolution operations into two steps. First, the neighbor vectors of a node are combined to generate a neighbor vector $\boldsymbol{n}_u$.

$$\boldsymbol{n}_u = \gamma(\{\mathrm{ReLU}(Q\boldsymbol{h}_v + \boldsymbol{q}) | v \in N(u)\}, \boldsymbol{\alpha}). \tag{4.45}$$

In the above equation, $Q\boldsymbol{h}_v + \boldsymbol{q}$ is to conduct a linear transformation of the neighbor vectors by using a single layer of neural network. γ is a vector-based pooling operation, which may be, for example, the average or weighted average of the neighbor vector on all its dimensions. Next, by concatenating the neighbor vector $\boldsymbol{n}_u$ and the self-vector $\boldsymbol{z}_u$ and feeding into a neural network, a new vector can be obtained.

$$\boldsymbol{z}_u^{\mathrm{NEW}} = \mathrm{ReLU}(\boldsymbol{W} \cdot \mathrm{CONCAT}(\boldsymbol{z}_u, \boldsymbol{n}_u) + \boldsymbol{w}) \tag{4.46}$$

$$\boldsymbol{z}_u^{\mathrm{NEW}} = \frac{\boldsymbol{z}_u^{\mathrm{NEW}}}{\|\boldsymbol{z}_u^{\mathrm{NEW}}\|_2}. \tag{4.47}$$

Similar to GCN, the convolution operations can be appended to combine the high-order neighbor information. On the other hand, GraphSage strictly controls the number of neighbors, $N(u)$ in the convolution operations, to avoid its exploding too fast. This not merely controls the size of the neighbor size but also simplifies the implementation complexity of the model with an improved use of memory. This makes the GraphSage algorithm trainable in mini-batch by using the gradient descent technique and resolves the issue that GCN cannot be trained scalably on a large graph.

(2) ClusterGCN Algorithm [11]

Given that it is difficult to train GCN on the whole graph, is it possible to partition the graph into sub-graphs by using the community detection method (e.g., METIS in [30]) and then train the sub-graphs with GCN? This is achieved by ClusterGCN. The ClusterGCN is naive yet efficient. It is advantageous in terms of low memory footprint. In each batch for training, it just loads the needed sub-graphs instead of the entire graph itself. It is also computationally efficient. For each node, the hidden states that have been calculated can be used for updating the objective function of the nodes. However, in GraphSage, only minorities of the sample nodes are capable of performing training against the objective function. To get the hidden states, the *k*-th order neighbors should be extended, but the training efficiency is quite low because there is no descent update for the *k*-th order neighbors.

(3) PPRGo Algorithm [5]

The limited scalability of graph neural network is due to its adding operations of neighbor nodes. Now that adding neighbor nodes is for getting useful information at the largest scale, is it possible to find the *k* most important nodes (these nodes are not limited to the directly connected ones) for each node in a graph as a fixed area so that the convolution operation is only performed in such area? Aleksandar Bojchevski et al. optimized the model for multiple iterations and proposed the PPRGo algorithm. It consists of two steps. In the first step, the algorithm performs a personalized PageRank (PPR) [28] on the node *u*, to find the *k* most important nodes to *u* as the neighborhood of the node. In the second step, the scores generated from PPR are used as the node weights of nodes in the neighborhood. By calculating the weighted sum of the latent factors of the neighbors, the new latent vector of node *u* can be obtained. The two steps of PPRGo are independent. In computation, the neighbors of all the nodes for pre-trained can be generated and fixed in the first step, and in the second step, the stochastic optimization can be done until a convergence is achieved.

One of the graph neural network use cases is the one that Rex Ying et al. proposed, which applied GraphSage into the recommender system of Pinterest [78]. The largest applied graph neural network, called PinSage, consists of 3 billion nodes and 18 billion edges. Compared to GraphSage, there are mainly two innovations in PinSage. One is that, for the neighbor sampling, PinSage conducts the random walk based on the graph to get the *k*-hop scores for the Top-*k* neighbors. When performing the pooling

operation on the neighbors, these scores can be used as the weights. By using the curriculum training philosophy, along with the training process, the weight of negative samples is increased. Meanwhile, to efficiently train the large-scale graph, there are three types of engineering optimizations performed in PinSage. First, the gradient descent training approach with mini-batch is used. For the nodes that are in a mini-batch, the local neighbors of that node are recursively sampled from a graph for training the *k*-layer network (e.g., to train a 2-layer graph neural network, second-order neighbors of the node are required). The training in the current iteration relies only on the information of the node and its edges, and the overall graph information is not needed. Second, by leveraging the design pattern of producer–consumer, preparation of the mini-batch training data that can be treated as a producer process only needs CPU-based computation, and the convolution operations on GPU devices can be treated as a consumer process. Since the CPU and the GPU devices can be parallelized, the producer and consumer setup can be pipelined for efficiency enhancement. Third, the graph neural network can be trained locally, and the implication can be performed on the global graph node latent vectors. This can be implemented by using the MapReduce-based distributed computing framework.

4.3.2 Graph Neural Network and Collaborative Filtering

GCMC Algorithm

Considering the user–item relationship an edge in the graph, the users and the items can be constructed as a bipartite graph. By doing this, the recommendation task can be formalized as a link prediction problem on the bipartite graph, and thus the graph model can be effectively applied in such use case. Rianne van den Berg et al. observed that the contemporary graph embedding models make use of the unsupervised learning approach to get the node features and use the features to train a downstream prediction model [2]. The two steps in this approach are independent so that it is lack of an end-to-end path that directly uses the features extracted from nodes for objective optimization. Rianne van den Berg et al. then proposed the Graph Convolutional Matrix Completion method, where CNN is used as extractor for node features with which an end-to-end training pipeline can be established. As shown in Fig. 4.14, given the user–item matrix, GCMC first builds a bipartite graph where the edges are tagged with classes to indicate

the ratings that users give to the items. Figure 4.14a depicts the original user–item rating matrix. Figure 4.14b is the bipartite graph constructed based on the rating matrix. Figure 4.14c illustrates that the content information for each node is passed to its neighbors via edges.

$$\boldsymbol{\mu}_{j\to i,r} = \frac{1}{c_{ij}}\boldsymbol{W}_r\boldsymbol{X}_j^v. \tag{4.48}$$

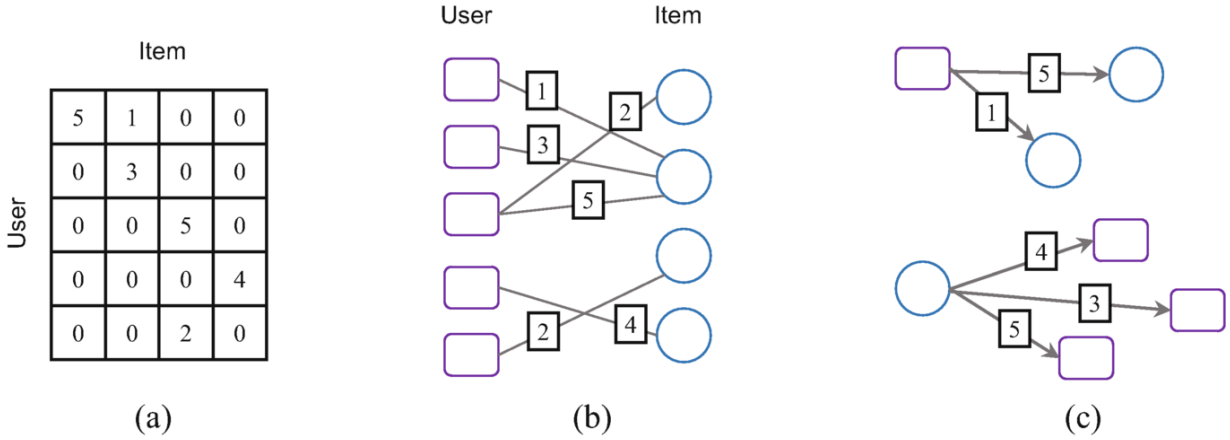


Fig. 4.14 Graph structure and message passing process for GCMC. (**a**) Original rating matrix. (**b**) Bipartite graph. (**c**) Message passing

In the above equation, $\boldsymbol{X}_j^v$ is the property vector of item v_j. $\boldsymbol{W}_r$ is the parameter matrix for data transformation tagged with r. c_{ij} is the regularization part. The degree of the receiving part $|N(u_i)|$ or the square root of the degrees of the two parts $\sqrt{|N(u_i)||N(v_j)|}$; $\boldsymbol{\mu}_{j\to i,r}$ can be used to represent the message that is passed from item node j to user node i via the edge tagged with r. User node i then combines all the information that is is received into a vector.

$$\boldsymbol{h}_i^u = \sigma\left(\text{accum}\left(\sum_{j\in N_1(u_i)}\boldsymbol{\mu}_{j\to i,1},\cdots,\sum_{j\in N_R(u_i)}\boldsymbol{\mu}_{j\to i,R}\right)\right). \tag{4.49}$$

In the above equation, $\text{accum}()$ is the function used for combination, e.g., vector concatenation, bit-wise addition, etc. After getting the vectors from neighbors, transformation with a neural network can be applied to get the latent vectors for the user nodes.

$$\boldsymbol{z}_i^u = \sigma(\boldsymbol{W}\boldsymbol{h}_i^u). \tag{4.50}$$

The latent factors of the item nodes can be obtained by doing the same. A bilinear function that delineates the preferences of users on items at each dimensionality of ratings is applied, after which an activation function of Softmax is used for normalization.

$$p(y_{ij} = r) = \frac{\mathrm{e}^{(\boldsymbol{z}_i^u)^\top \boldsymbol{Q}_r \boldsymbol{z}_j^v}}{\sum_{s=1}^{R} \mathrm{e}^{(\boldsymbol{z}_i^u)^\top \boldsymbol{Q}_s \boldsymbol{z}_j^v}}. \tag{4.51}$$

In the above equation, $\boldsymbol{Q}_r$ is the parameters of the bilinear function corresponding to the rating *r*. The final prediction score is the expected value of all the predicted ratings.

$$\hat{y}_{ij} = \mathbb{E}_{p(y_{ij}=r)}[r] = \sum_{s=1}^{R} r \cdot p(y_{ij} = r). \tag{4.52}$$

Considering that every rating value r corresponds to a set of model parameters $\boldsymbol{W}_r$, $\boldsymbol{Q}_r$, when the user data are sparse, some parameters cannot be fully optimized. Therefore, a parameter sharing technique is introduced, to aggregate the low-level rating parameters $\boldsymbol{T}_s$ for the parameter of W_r.

$$\boldsymbol{W}_r = \sum_{s=1}^{r} \boldsymbol{T}_s. \tag{4.53}$$

For $\boldsymbol{Q}_r$, it can be obtained by learning the parameter coefficients a_{rs} from n_b basic parameter matrices $\boldsymbol{P}_s$.

$$\boldsymbol{Q}_r = \sum_{s=1}^{n_b} a_{rs} \boldsymbol{P}_s. \tag{4.54}$$

NGCF Algorithm

Xiang Wang proposed the NGCF algorithm at SIGIR'19 [67]. It is a collaborative filtering algorithm that is enhanced by graph neural network. The conventional model-based collaborative filtering algorithm such as matrix factorization algorithm obtains the latent factors from the encoding of the user or item IDs (or properties). Such encoding process does not combine the collaborative signal in user–item interactions, and thus the latent factors generated cannot fully describe the collaborative filtering impact in the data. The high-order connectivity of a node can be used to

exploit the user–item interaction information. For example, a path <user a, item b, user c> on the graph reflects the fact that user a and user c are similar because both of them like item b. Another path of <user a, item b, user c, item d> indicates that item d may be liked by user a because the similar users a and c like item d, and this is exactly the idea of collaborative filtering. Therefore, NGCF attempts to use the graph neural network to introduce the high-order connectivity into the encoding process, to get an even more powerful representation of latent factors. The overall structure of NGCF is similar to GCMC, which also passes messages via edges to neighbors and combines the information of all the neighbors. However, it is different from GCMC in terms of the passing and the combining operations. Without loss of generality, considering merely the collaborative filtering methods where users and items do not have property information but the ID latent factors, the following can be derived.

$$\boldsymbol{E} = [\boldsymbol{e}_{u_1}, \cdots, \boldsymbol{e}_{u_N}, \boldsymbol{e}_{i_1}, \cdots, \boldsymbol{e}_{i_M}]. \tag{4.55}$$

The message that is passed from item i to user u $\boldsymbol{\mu}_{i\to u}$ is

$$\boldsymbol{\mu}_{i\to u} = \frac{1}{\sqrt{|N(u)||N(i)|}}(\boldsymbol{W}_1\boldsymbol{e}_i + \boldsymbol{W}_2(\boldsymbol{e}_i \odot \boldsymbol{e}_u)). \tag{4.56}$$

In the above equation, $\odot$ represents the new vector that is the inner product of the two vectors. The message that user passes to itself is $\boldsymbol{\mu}_{u\to u} = \boldsymbol{W}_1\boldsymbol{e}_u$. The connectivity information of user u can be obtained by the operations as below.

$$\boldsymbol{e}_u^{(1)} = \text{LeakyReLU}\left(\boldsymbol{\mu}_{u\to u} + \sum_{i\in N(u)} \boldsymbol{\mu}_{i\to u}\right). \tag{4.57}$$

This process can be appended to get the even higher-order connectivity information.

$$\boldsymbol{\mu}_{i\to u}^{(l)} = \frac{1}{\sqrt{|N(u)||N(i)|}}\left(\boldsymbol{W}_1^{(l)}\boldsymbol{e}_i^{(l-1)} + \boldsymbol{W}_2^{(l)}\left(\boldsymbol{e}_i^{(l)} \odot \boldsymbol{e}_u^{(l-1)}\right)\right) \tag{4.58}$$

$$\boldsymbol{\mu}_{u\to u}^{l} = \boldsymbol{W}_1^{(l)}\boldsymbol{e}_u^{(l-1)} \tag{4.59}$$

$$\boldsymbol{e}_u^{(l)} = \text{LeakyReLU}\left(\boldsymbol{\mu}_{u\to u}^{(l)} + \sum_{i\in N(u)} \boldsymbol{\mu}_{i\to u}^{(l)}\right). \tag{4.60}$$

The modeling process for items is similar. The neural network with different depths corresponds to information with different breadths. NGCF combines the vectors to form a long one as the user and item latent factors.

$$\boldsymbol{e}_u^* = \boldsymbol{e}_u^{(0)} \oplus \boldsymbol{e}_u^{(1)} \cdots \oplus \boldsymbol{e}_u^{(L)}, \boldsymbol{e}_i^* = \boldsymbol{e}_i^{(0)} \oplus \boldsymbol{e}_i^{(1)} \cdots \oplus \boldsymbol{e}_i^{(L)}. \tag{4.61}$$

The inner product is used as the predicted interest scores for user toward item.

$$\hat{y}(u,i) = \boldsymbol{e}_u^{*\top} \boldsymbol{e}_i^*. \tag{4.62}$$

The algorithm is optimized by maximizing the pairwise logloss. Making the predicted value of positive samples toward (u,i) be larger than that of the negative samples as much as possible, the loss function can be derived as below.

$$\text{Loss} = \sum_{(u,i,j)\in O} -\ln \sigma(\hat{y}(u,i) - \hat{y}(u,j)) + \lambda\Theta_2^2. \tag{4.63}$$

In the above equation, 0 indicates the tuples in the training set, where *u* is user, *i* and *j* are two different items, and user *u* has rating on *i* but not *j*, σ is the Sigmoid function, and Θ is the set of the model parameters.

LightGCN Algorithm

While GCN is popular, it is merely an algorithm that is used for classification on graph. Although researchers applied GCN on the recommender system (e.g., NGCF) and made success, the internal structure of GCN and its impact on recommendation tasks had not been thoroughly studied. With exhaustive experiment, Xiangnan He et al. [21] found that the two key components in GCN, i.e., feature transformation and non-linear activation function, do not help in the recommendation tasks. Therefore, they simplify the NGCF algorithm by pruning the unnecessary components. The new algorithm is named as LightGCN. The combination operation on the connectivity information is simplified to weighted average of vectors and the activate function is removed. That is,

$$\boldsymbol{e}_u^{(k+1)} = \sum_{i \in N(u)} \frac{1}{\sqrt{|N(u)||N(i)|}} \boldsymbol{e}_i^{(k)} \tag{4.64}$$

$$\boldsymbol{e}_i^{(k+1)} = \sum_{u \in N(i)} \frac{1}{\sqrt{|N(u)||N(i)|}} \boldsymbol{e}_u^{(k)}. \tag{4.65}$$

The final latent factors for user and item nodes are not the concatenation of the ones generated from each layer. Instead, the average vector is used.

$$\boldsymbol{e}_u^* = \sum_{k=0}^{L} \frac{1}{L+1} \boldsymbol{e}_u^{(k)}, \boldsymbol{e}_i^* = \sum_{k=0}^{L} \frac{1}{L+1} \boldsymbol{e}_i^{(k)}. \tag{4.66}$$

By experimenting on the public dataset, Xiannan He et al. surprisingly found that the performance of LightGCN improves that of NGCF by 16.5% on average. Since there is no feature transformation and non-linear activate function, its computation process can be conveniently formalized as matrix operations. Letting the user–item rating matrix become $\boldsymbol{R} \in \mathbb{R}^{M \times N}$, where *M* and *N* represent the number of users and items, respectively. The adjacent matrix of the user–item bipartite graph can be generated as below.

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{R} \\ \boldsymbol{R}^\top & \boldsymbol{0} \end{pmatrix}. \tag{4.67}$$

Letting the matrix formed by using the latent factors from the user and the item IDs be $\boldsymbol{E}^{(0)} \in \mathbb{R}^{(M+N) \times D}$, the convolution operation in LightGCN can be formed as

$$\boldsymbol{E}^{(k+1)} = (\boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}) \boldsymbol{E}^{(k)} \stackrel{\text{def}}{=} \tilde{\boldsymbol{A}} \boldsymbol{E}^{(k)}. \tag{4.68}$$

$\boldsymbol{D}$ is the diagonal matrix that is generated from degree of each node in $\boldsymbol{A}$. The final latent factors for the nodes are

$$\boldsymbol{E}^* = \alpha_0 \boldsymbol{E}^{(0)} + \alpha_1 \tilde{\boldsymbol{A}} \boldsymbol{E}^{(1)} + \alpha_2 \tilde{\boldsymbol{A}}^2 \boldsymbol{E}^{(2)} + \cdots + \alpha_L \tilde{\boldsymbol{A}}^L \boldsymbol{E}^{(L)}. \tag{4.69}$$

In the above equation, α_* is the weight coefficient. In LightGCN, it is set to $\frac{1}{L+1}$. From the perspective of matrix operation, LightGCN has a similar structure to RandNE [82] except that ID embedding matrix of the latter cannot be learnt.

4.3.3 Graph Neural Network and Social Recommendation

Social recommendation leverages the social media network that users participate in for enhancing the recommendation precision of a recommender system. The social connections of a user, e.g., friends, schoolfellows, colleagues, etc., contain rich and close group relationship. One the one hand, the users that are in the same social community may have the common interest. On the other hand, users may be influenced by the others that have a close relationship or are trustworthy such that they may behave in a similar way. A good use of the social network information not only helps predict user interests but also helps dealing with the cold-start problem for new users. However, how to use the user–user social network graph and the user–item behavior graph is a big challenge.

GraphRec Algorithm

Wenqin Fan et al. proposed the GraphRec algorithm. It generates a user vector by concatenating the two vectors that are produced based on the item space and the user social network space. Specifically, the item space vector is represented as $\boldsymbol{h}_i^I$, which indicates the vector generated by modeling the user–item historic behavior.

$$\boldsymbol{h}_i^I = \sigma\left(\boldsymbol{W} \cdot \left\{\sum_{\alpha \in C(i)} \alpha_{ia} \boldsymbol{x}_{ia}\right\} + \boldsymbol{b}\right). \tag{4.70}$$

In the above equation, $C(i)$ is the item set that user u_i has interacted. $\boldsymbol{x}_{ia}$ is the latent factor of the items v_a that user u_i likes. The ratings that user has on the items are discrete values, e.g., $r \in \{1, 2, 3, 4, 5\}$. A latent factor $\boldsymbol{e}_r$ is assigned to each rating for updating the original vector representation of the item $\boldsymbol{q}_a$ $\boldsymbol{x}_{ia}$, and this is the adjusted vector representation for the rating *r* given to item v_a by the users.

$$\boldsymbol{x}_{ia} = g_v([\boldsymbol{q}_a \oplus \boldsymbol{e}_r]), \tag{4.71}$$

where $g_v()$ is an MLP. α_{ia} is the weight coefficients for the interaction set between item v_a and user u_i.

$$\alpha_{ia}^* = \boldsymbol{w}_2^\top \cdot \sigma(\boldsymbol{W}_1 \cdot [\boldsymbol{x}_{ia} \oplus \boldsymbol{p}_i] + \boldsymbol{b}_1) + b_2 \tag{4.72}$$

$$\alpha_{ia} = \frac{\exp(\alpha_{ia}^*)}{\sum_{a \in C(i)} \exp(\alpha_{ia}^*)}. \tag{4.73}$$

$\boldsymbol{p}_i$ indicates the latent factor for user u_i. The user vector based on the social network space is represented as $\boldsymbol{h}_i^S$, and it is formed by combining the vectors from the user's neighbors.

$$\boldsymbol{h}_i^S = \sigma\left(\boldsymbol{W} \cdot \left\{\sum_{o \in N(i)} \beta_{io} \boldsymbol{h}_o^I\right\} + \boldsymbol{b}\right) \tag{4.74}$$

$$\beta_{io}^* = \boldsymbol{w}_2^\top \cdot \sigma(\boldsymbol{W}_1 \cdot [\boldsymbol{h}_o^I \oplus \boldsymbol{p}_i] + \boldsymbol{b}_1) + b_2 \tag{4.75}$$

$$\beta_{io} = \frac{\exp(\beta_{io}^*)}{\sum_{o \in N(i)} \exp(\beta_{io}^*)}. \tag{4.76}$$

Considering the item space and the social network space reflect the user preferences at different aspects, to obtain the final user vector $\boldsymbol{h}_i$, the two vectors are concatenated and fed into the MLP layer.

$$\begin{aligned} \boldsymbol{c}_1 &= [\boldsymbol{h}_i^I \oplus \boldsymbol{h}_i^S] \\ \boldsymbol{c}_2 &= \sigma(\boldsymbol{W}_2 \cdot \boldsymbol{c}_1 + \boldsymbol{b}_2) \\ &\cdots \\ \boldsymbol{h}_i &= \sigma(\boldsymbol{W}_l \cdot \boldsymbol{c}_{l-1} + \boldsymbol{b}_l). \end{aligned} \tag{4.77}$$

Similar to the user latent factors, $\boldsymbol{h}_i^I$, the items can also be represented by all the users they have interacted with. To be more specific, the original vector of item $\boldsymbol{p}_t$ can be adjusted by the rating vectors.

$$\boldsymbol{f}_{jt} = \mathrm{MLP}_u([\boldsymbol{p}_t \oplus \boldsymbol{e}_r]). \tag{4.78}$$

The item vector for v_j becomes $\boldsymbol{q}_j$ from $\boldsymbol{z}_j$.

$$\boldsymbol{z}_j = \sigma\left(\boldsymbol{W} \cdot \left\{\sum_{t \in B(j)} \mu_{jt} \boldsymbol{f}_{jt}\right\} + \boldsymbol{b}\right) \tag{4.79}$$

$$\mu_{jt}^* = \boldsymbol{w}_2^\top \cdot \sigma(\boldsymbol{W}_1 \cdot [\boldsymbol{f}_{jt} \oplus \boldsymbol{q}_j] + \boldsymbol{b}_1) + b_2 \tag{4.80}$$

$$\mu_{jt} = \frac{\exp(\mu_{jt}^*)}{\sum_{t \in B(j)} \exp(\mu_{jt}^*)}. \tag{4.81}$$

Finally, the rating predictions generated from the MLP layer with the input of the concatenated user and item vectors.

$$\begin{aligned} \boldsymbol{g}_1 &= [\boldsymbol{h}_i \oplus \boldsymbol{z}_j] \\ \boldsymbol{g}_2 &= \sigma(\boldsymbol{W}_2 \cdot \boldsymbol{g}_1 + \boldsymbol{b}_2) \\ &\cdots \\ \boldsymbol{g}_l &= \sigma(\boldsymbol{W}_l \cdot \boldsymbol{g}_{l-1} + \boldsymbol{b}_l) \\ r'_{ij} &= \boldsymbol{w}^\top \cdot \boldsymbol{g}_l. \end{aligned} \tag{4.82}$$

DiffNet Algorithm

GraphRec combines the first-order neighbor information on a social network graph to get an enhanced representation of user state. Le Wu et al. proposed Diffusion Neural Network (DiffNet) by leveraging the idea of social influence propagation process to model the user interests in a social network graph as layer-wise influence diffusion. This method cannot merely fuse the first-order information of neighbors, but also learn about how to recursively combine the information of the neighbors of the neighbor in order to take in even higher-order information. To be specific, using $\boldsymbol{P} \in \mathbb{R}^{D \times M}$ and $\boldsymbol{Q} \in \mathbb{R}^{D \times N}$ to be the latent factors of the user and item IDs, the hidden states of user a and item i are both generated from a fully connected layer with the input of the concatenation of ID vector and the property features.

$$\boldsymbol{h}_a^0 = g(\boldsymbol{W}^0 \cdot [\boldsymbol{x}_a \oplus \boldsymbol{p}_a] + \boldsymbol{b}_0) \tag{4.83}$$

$$\boldsymbol{v}_i = \sigma(\boldsymbol{F} \cdot [\boldsymbol{y}_i \oplus \boldsymbol{q}_i] + \boldsymbol{b}). \tag{4.84}$$

The item vector does not need the influence diffusion. For user vector, K influence diffusion is needed to simulate the influencing process that the neighbors have on the user. To use $\boldsymbol{h}_a^k$ to indicate the state of the user *a* after the information is propagated for k times. The $k+1$-th propagation process is

$$\boldsymbol{h}_{S_a}^{k+1} = \mathrm{Pool}(\boldsymbol{h}_b^k | b \in S_a) \tag{4.85}$$

$$\boldsymbol{h}_a^{k+1} = s^{(k+1)}(\boldsymbol{W}^k \cdot [\boldsymbol{h}_{S_a}^{k+1} \oplus \boldsymbol{h}_a^k]). \tag{4.86}$$

In the above equation, Pool is the pooling operation that can be either an average combination or a bit-wise max operation. $s^{(k+1)}$ is a non-linear transformation function. The final user vector $\boldsymbol{u}_a$ is the combination of output from the *K*-th diffusion network and the state of the historic set of the items that user has interacted R_a.

$$\boldsymbol{u}_a = \boldsymbol{h}_a^K + \sum_{i \in R_a} \frac{\boldsymbol{v}_i}{|R_a|}. \tag{4.87}$$

The users' interest on items is modeled as the inner product of the two vectors.

$$\hat{r}_{ai} = \boldsymbol{v}_i^\top \boldsymbol{u}_a. \tag{4.88}$$

The optimization objective is the BPR loss.

$$L(R, \hat{R}) = \sum_{a=1}^{M} \sum_{(i,j) \in D_a} \sigma(\hat{r}_{ai} - \hat{r}_{aj}) + \lambda \theta_1^2. \tag{4.89}$$

In the equation, $\theta_1 = [\boldsymbol{P}, \boldsymbol{Q}]$ is the L_2 regularization form for the latent factors for the user and item IDs.

In the following year when DiffNet was published, Le Wu et al. proposed an improved version of DiffNet that is called DiffNet++ [71]. The main framework of DiffNet++ is the still based on DiffNet. There are two improvements made in DiffNet++. One is that both the user–user social network graph and the user–item interaction bipartite graph are considered for influence propagation, to get the graph-based user and item representations. When combining the neighbor information, an attention mechanism is used for learning the weighted pooling operations. To be specific, the user representation on the user–item bipartite graph is

$$\tilde{\boldsymbol{v}}_i^{k+1} = \sum_{a \in R_i} \eta_{ia}^{k+1} \boldsymbol{u}_a^k \tag{4.90}$$

$$\boldsymbol{v}_i^{k+1} = \tilde{\boldsymbol{v}}_i^{k+1} + \boldsymbol{v}_i^k. \tag{4.91}$$

R_i is the user set where the users have interacted with item i . η_{ia}^{k+1} is the weight coefficient that indicates how significant the user a is toward item i . It can be computed as below.

$$\tilde{\eta}_{ia}^{k+1} = \mathrm{MLP}_1([\boldsymbol{v}_i^k \oplus \boldsymbol{u}_a^k]) \tag{4.92}$$

$$\eta_{ia}^{k+1} = \frac{\exp(\tilde{\eta}_{ia}^{k+1})}{\sum_{b \in R_i} \tilde{\eta}_{ib}^{k+1}}. \tag{4.93}$$

The user representation does not merely combine the neighbors on the social network graph but also the neighbors on the user–item bipartite graph.

$$\boldsymbol{u}_a^{k+1} = \boldsymbol{u}_a^k + (\gamma_{a1}^{k+1} \tilde{\boldsymbol{p}}_a^{k+1} + \gamma_{a2}^{k+1} \tilde{\boldsymbol{q}}_a^{k+1}) \tag{4.94}$$

$$\tilde{\boldsymbol{p}}_a^{k+1} = \sum_{b \in S_a} \alpha_{ab}^{k+1} \boldsymbol{u}_b^k, \quad \tilde{\boldsymbol{q}}_a^{k+1} = \sum_{i \in R_a} \beta_{ai}^{k+1} \boldsymbol{v}_i. \tag{4.95}$$

Similarly, the weights of the combination can be predicted from the MLP layer.

$$\alpha_{ab}^{k+1} = \mathrm{MLP}_2([\boldsymbol{u}_a^k \oplus \boldsymbol{u}_b^k]) \tag{4.96}$$

$$\beta_{ai}^{k+1} = \mathrm{MLP}_3([\boldsymbol{u}_a^k \oplus \boldsymbol{v}_i^k]). \tag{4.97}$$

Lastly, for users and items, the final vector representation can be obtained by concatenating the hidden states in each layer.

$$\boldsymbol{u}_a^* = [\boldsymbol{u}_a^0 \oplus \boldsymbol{u}_a^1 \oplus \cdots \oplus \boldsymbol{u}_a^K], \quad \boldsymbol{v}_i^* = [\boldsymbol{v}_i^0 \oplus \boldsymbol{v}_i^1 \oplus \cdots \oplus \boldsymbol{v}_i^K]. \tag{4.98}$$

4.4 Sequential Recommender Systems

Sequential recommendation is a type of recommendation paradigm that recommends relevant items to users by modeling patterns of user behavior and item interactions over time. In recommender systems, there are two main entities: users and items, both of which involve multiple interactions over time, such as browsing, clicking, and purchasing behaviors. Sequential recommender systems arrange these interaction behaviors in chronological order, utilizing various modeling methods to uncover sequential patterns and support the recommendation of one or multiple items for the next moment.

This section first introduces the research motivation and mathematical definitions of sequential recommendation. It then proceeds to classify sequential recommendation techniques in the order of their publication, discussing three main categories of these techniques. Subsequently, the focus shifts to an in-depth explanation of sequential recommendation algorithms based on deep learning methods. Finally, several cutting-edge topics that have garnered significant attention in both academia and industry are presented.

4.4.1 Motivation, Definition, and Classification of Sequential Recommendation

In the process of regular online browsing, users engage in various browsing behaviors. As shown in Fig. 4.15, a user's browsing records on an e-commerce platform are arranged in chronological order. The user's recent history on the platform includes a sequence of different interaction behaviors such as "browsing", "searching", and "purchasing". The items of interest to the user comprise "mobile phones", "headphones", "phone screen protectors", and other related products, indicating that this user is currently in the process of selecting mobile communication devices and related items.

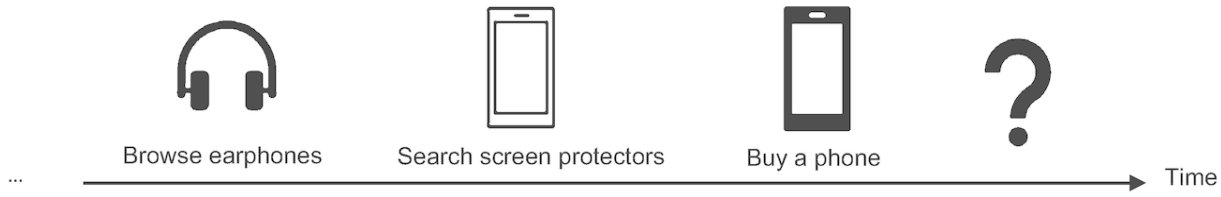


Fig. 4.15 Example of user's historical behavior sequence

Based on the sequence of user's recent browsing history, the e-commerce platform can leverage the information to recommend products like "phone cases" when the user visits the platform next time. Such a recommendation is more likely to elicit positive user feedback, such as "clicking to view" or "making a purchase conversion".

Through this example, readers should be able to understand the scenarios in which sequential recommendation occurs, which involves recommending items based on users' dynamically changing historical behavior sequences for future moments. In real life, people's interests often change over time, and at the same time, the behavior sequences driven by user interests can be diverse. For instance, before a trip, users might be interested in hotels, flight

tickets, and travel equipment; during festivals like Chinese New Year, festive-themed items tend to be of particular interest to users. Therefore, in different time periods, for different users and different contexts, user behavior sequences exhibit various forms and variations. Readers can consider whether their online shopping, web content browsing, and other behaviors also demonstrate continuous and dynamic characteristics in their daily lives.

Since the late 20th century, the Internet has evolved, giving rise to numerous online service platforms, such as e-commerce giants such as Amazon and Alibaba, news media platforms such as Toutiao, and social media platforms such as Meta and Sina Weibo. These platforms have often moved past the phase of rapid user growth ("incremental development") and entered a period of "stock growth", where new user growth has slowed down, and the focus lies in providing better services and experiences to the existing user base. These platforms' existing users continuously engage in various types of interactions. Ren et al. [47] conducted a study on user behavior data on Alibaba's e-commerce platform "Taobao" during the period from April to September 2018. The data show the percentage of users with different interaction frequencies on the platform during that six-month period, specifically focusing on the interactions between "adding items to the shopping cart" and the final purchase. From Table 4.1, it can be observed that a significant portion of users on "Taobao" had a high number of interactions, with more than half of the users having over 200 interactions, indicating that their behavior sequence length exceeded 200 times within the six months. Consequently, exploring how to model the dynamic changes in these users' interests to facilitate personalized recommendations during their next visit has become a vital research direction for platforms such as "Taobao".

Table 4.1 User behavior statistics on Alibaba E-commerce Platform "Taobao" from April to September 2018 [47]

Length of behavior sequence	**Less than 200 times**	**Less than 1,000 times**	**Less than 3,000 times**
Percentage of users	47.57%	77.08%	92.73%

The recommendation and ranking algorithms based on users' historical behavior sequences have become essential in various major online platforms.

According to a paper from the "Alimama" advertising platform, their algorithm called Deep Interest Network (DIN) [87], which was the first to utilize user historical behavior sequences for modeling, resulted in a 10% increase in user click-through rate and a 3.8% increase in platform revenue after its launch. Subsequently, the adoption of sequence modeling techniques in the Deep Interest Evolution Network (DIEN) [86] led to another over 10% increase in user click-through rate and a 9.7% increase in platform revenue. Such rapid growth in user click-through rate and platform revenue was not solely due to a massive influx of new users; it was primarily the result of research efforts to improve the modeling of user historical behavior data and the enhancement of algorithm models. This demonstrates the importance and promising prospects of modeling and recommendation algorithms based on users' historical behavior sequences.

In the context of sequential recommendation, the dataset is organized in time order as either user *u*'s evaluation scores for item *v* (e.g., R_t) [70] or event behaviors (e.g., y_t) such as clicks, purchases, and conversions [29]. For the sake of clarity, this section will use "click" behavior as an example to illustrate the concepts.

Mathematically, sequential recommendation can be defined as a recommendation task that incorporates three pieces of information: the current time *t*, the user-side representation vector $\boldsymbol{u}_t$ for a given user *u*, the item-side representation vector $\boldsymbol{v}_t$ for a specific item *v*, and the context information representation vector $\boldsymbol{c}_t$.

The goal of the recommendation system is to predict the user *u*'s rating score r_t for item *v* or the click probability p_t as follows. For rating prediction:

$$r_t = f(\boldsymbol{u}_t, \boldsymbol{v}_t, \boldsymbol{c}_t; \boldsymbol{\Phi}). \tag{4.99}$$

For click probability prediction $p_t = \Pr(y_t = 1 | \boldsymbol{u}, \boldsymbol{v}, t)$:

$$p_t = f(\boldsymbol{u}_t, \boldsymbol{v}_t, \boldsymbol{c}_t; \boldsymbol{\Phi}). \tag{4.100}$$

In the above equations, $\boldsymbol{\Phi}$ represents the parameters of the prediction function $f(\cdot)$, and it is typically trained using a loss function based on the specific prediction target.

$$L_{\boldsymbol{\Phi}} = \begin{cases} \frac{1}{2}(R_t - r_t)^2, & \text{for rating prediction,} \\ -y_t \log(p_t) - (1 - y_t) \log(1 - p_t), & \text{for click probability prediction.} \end{cases} \tag{4.101}$$

Here, R_t is the actual rating given by the user (if rating prediction is the target), y_t is the binary label indicating whether the user clicked on item *v* at time *t*, and $f(\cdot)$ is the prediction function that takes user, item, and context representations as inputs to produce the prediction. The training process involves minimizing the loss function with respect to the parameters Φ to improve the accuracy of the recommendation model in predicting user ratings or click probabilities.

For the user-side representation $\boldsymbol{u}_t$, sequential recommendation algorithms integrate the interaction set $G_t(u) = \{v_j | y_j(u, v_j) = 1, j < t\}$ of all items v_j that user *u* has interacted with or rated before time *t*. Here, the interaction set $G_t(u)$ represents all items with which user *u* has had interactions before time *t*. By modeling the user's past interaction set, sequential recommendation can more accurately capture changes in user interests and depict the relationships between user behaviors, thus achieving more precise recommendation results.

There are three categories of sequential recommendation based on different scenarios, as shown in Fig. 4.16.

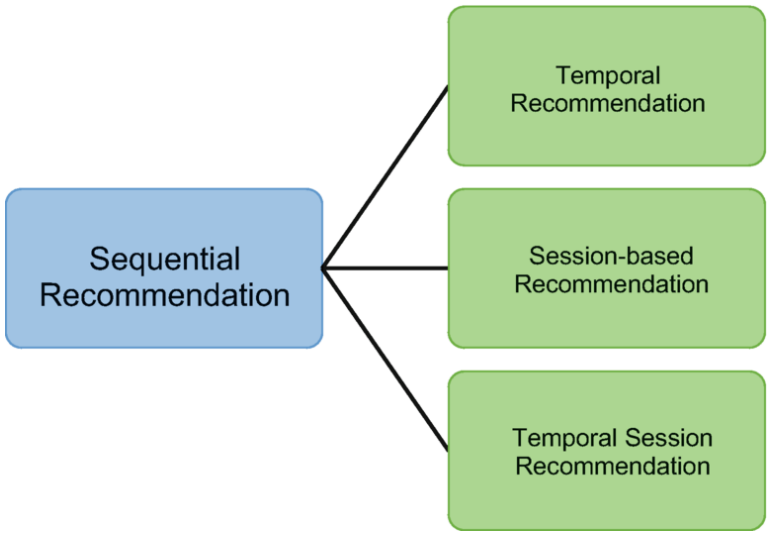


Fig. 4.16 Classification of sequential recommendation

Temporal recommendation refers to recommending the next item [20, 40] or the next set of items [48, 63] of interest for a specific user *u* based on their historical behavior sequence up to time *t*. In the mathematical definition provided earlier, the recommendation target is defined as the "next" item recommendation. If recommending a "next set" of items, the definition of *v*

should be adapted to represent a “set of items” while keeping the other definitions unchanged. In temporal recommendation, users may not be logged in, and in such cases, providing recommendations based on an anonymous user’s short-term session within a single session is referred to as session-based recommendation. In session-based recommendation, the length of the sequence is generally short, and it may not be possible to associate with the user’s longer-term historical interaction behaviors. The third category involves combining the first two types of sequences. The system considers both the short-term browsing behaviors in the current session and the long-term historical behaviors of logged-in users to provide comprehensive recommendations.

Overall, these three types of sequential recommendations cater to different scenarios and user engagement levels, enabling the system to adapt and deliver more relevant and accurate recommendations.

The sequential recommendation scenario presents three challenges that make it difficult to address using traditional recommendation algorithms:

- Dynamic nature of user interests: Users’ interests change and shift over time. Their preferences and behavior patterns evolve, making it necessary for the modeling methods to adapt to this dynamic nature.
- Dependency between different items in the user behavior sequence: There are interdependencies between different items in a user’s behavior sequence. For example, there may be sequential dependencies, such as “purchasing a mobile phone” being followed by “searching for phone screen protectors”. Modeling such dependencies is essential for accurate recommendations.
- Context modeling: Sequential recommendation often involves unique contextual relationships. Users’ feedback and behaviors may vary in different usage contexts, which necessitates considering contextual information for effective recommendation.

Traditional recommendation algorithms, such as matrix factorization algorithms [33] and wide and deep network algorithms [10], typically treat user historical behavior as static information. Even when considering user behavior sequences, they often rely on complex handcrafted feature engineering to address the challenges mentioned above. However, these approaches are limited in their ability to fully resolve the issues unique to sequential recommendation. Consequently, research on sequential

recommendation algorithms has emerged to tackle these challenges more effectively.

4.4.2 Classification of Sequential Recommendation Algorithms

As shown in Fig. 4.17, sequential recommendation algorithms can be classified into four categories. In the following, this section will provide a brief overview of each category.

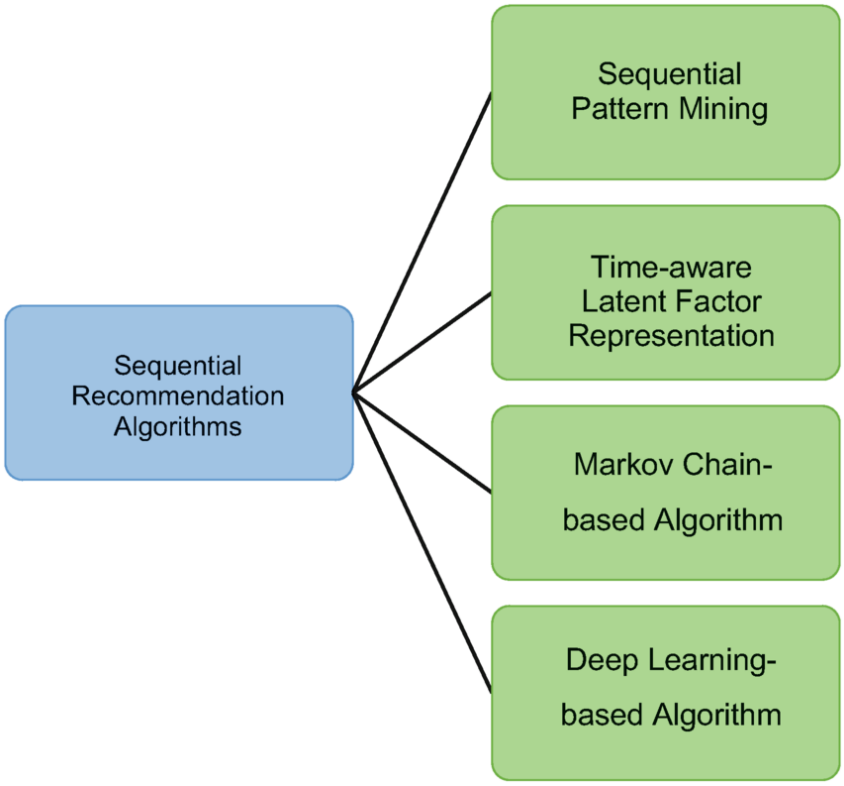


Fig. 4.17 Classification of sequential recommendation algorithms

Sequential Pattern Mining

Sequential pattern mining (SPM) was proposed by Yap et al. in 2012 [76]. Before that, Agrawal et al. introduced the concept of sequential pattern mining [1] as a method to recommend the next item based on mining sequential patterns. The underlying assumption of this technique is that if many users browse item v_i and then subsequently browse item v_j, it is reasonable for the recommendation system to suggest item v_j to users who have viewed item v_i. Figure 4.18 illustrates the process of the sequential pattern mining algorithm.

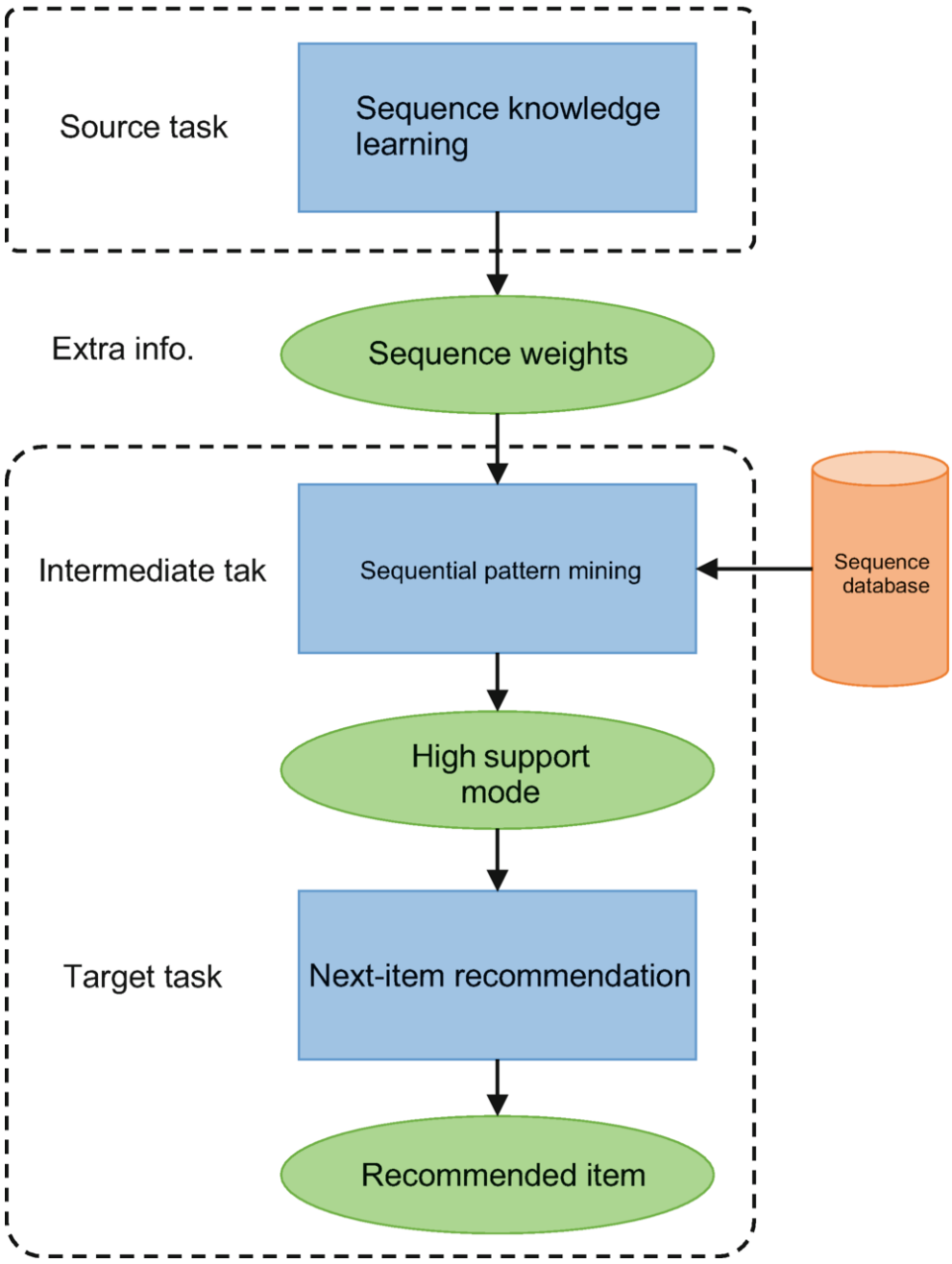


Fig. 4.18 Process of sequence pattern mining algorithms

Yap et al. believed that the traditional approach had limitations in achieving personalized recommendations. To address this, they built upon traditional sequential pattern mining and utilized the results of pattern mined on a specific user's browsing history to calculate support and predictive power. These metrics were then used for personalized sequential recommendation. This category of methods uses traditional sequential pattern mining algorithms, which require significant computational resources

and rule design. Additionally, they face challenges in modeling complex sequential relationships. Nonetheless, this was an early exploration in the field of sequential recommendation algorithms.

Latent Factor Representation

The second category of methods is the time-aware latent factor representation method. These methods are closely related to traditional collaborative filtering algorithms and factorization machine algorithms. They use latent vector representations to model user and item features and perform first-order and second-order operations to predict user ratings or click probabilities.

In 2009, Koren [32] first proposed a collaborative filtering algorithm that models user interest transfer and changes over time. By introducing the time factor into the factor model, Koren's timeSVD++ algorithm extends the original SVD++ algorithm. It incorporates linear and spline functions to comprehensively model the temporal transfer of user global interests and specific interests at a given time. The formula for timeSVD++ is as follows:

$$\hat{r}_{uv}(t) = \mu + b_v(t) + b_u(t) + \boldsymbol{q}_v^\top \left[\boldsymbol{p}_u(t) + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} \boldsymbol{y}_j \right], \quad \boldsymbol{q}_v \in \mathbb{R}^k. \tag{4.102}$$

In the formula, μ represents the average rating, and $b_u(t)$ and $b_v(t)$ are the time-varying user bias and item bias, respectively. $\boldsymbol{q}_v$ denotes the item feature vector, which is assumed to undergo minimal changes over time (as item features are less likely to change significantly with time). $\boldsymbol{p}_u(t)$ represents the time-varying user feature, $R(u)$ is the set containing items rated by user *u*, and $\boldsymbol{y}_j$ represents the item factor features. This formula decomposes user ratings into different components, while considering time-varying user features and biases, making it capable of modeling temporal features.

This algorithm, by explicitly modeling the changes in user interests, has significantly improved performance compared to traditional methods. Subsequently, there have been related works that incorporate time or user behavior sequences into factorization-based representation methods [8, 43]. In a research work by Tong Chen et al. [8], it was pointed out that traditional factorization methods often overlook the sequential information in user interactions. Even though Rajiv Pasricha et al. [43] had already considered

sequence features, they only took into account the influence of the most recent item, which could lead to information biases and incorrect recommendation results.

In response, Tong Chen proposed a Sequence-Aware Factorization Machine (SAFM) algorithm [8] for predicting user ratings or click probabilities in the next time step. As shown in Fig. 4.19, the features from both the user and item sides are divided into static view features and dynamic view features. The paper employed interactions between static view features and a combination of cross-view feature interactions and dynamic view feature interactions, implemented using a neural network structure based on self-attention mechanisms. Additionally, the paper masked the input sequence based on the order of user–item interactions to ensure the correct usage of the sequence during modeling and avoid information leakage.

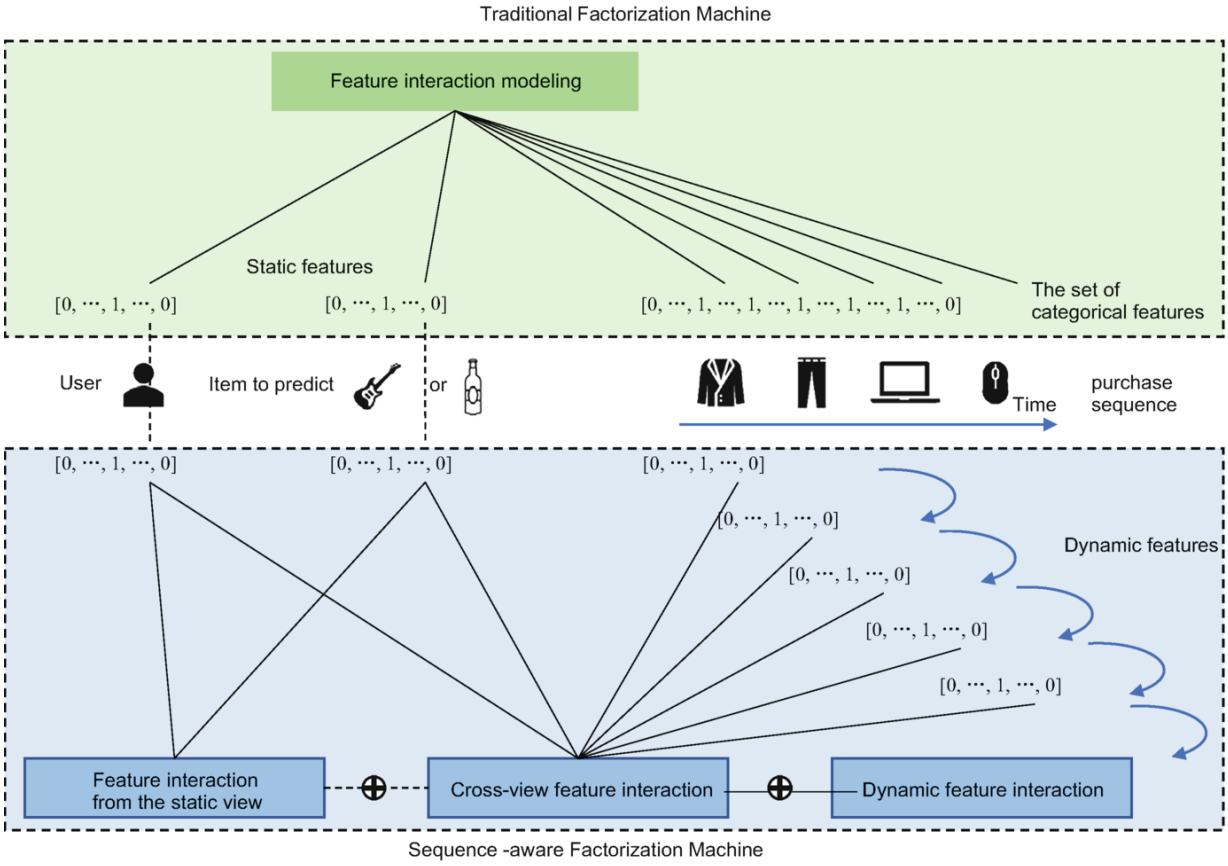


Fig. 4.19 Sequence-Aware Factorization Machine model for temporal prediction analysis [8]

The experimental comparisons in the paper demonstrated SAFM's superiority over traditional factorization machine methods and other algorithms, thereby showcasing the advantage of using sequence information for factor modeling in sequence prediction tasks.

Markov Chain-Based Sequential Recommendation

The third category is the Markov chain-based sequential modeling method. The Markov Chain (MC) is a well-known approach used for modeling state probability distributions and state transition probabilities. The core idea is to model the user's purchase or click behavior at time *t* as the conditional probability given the purchase behavior at time $(t-1)$:

$$p(B_t|B_{t-1}). \tag{4.103}$$

Here, $B_t \in 2^{|I|}$ represents the user's purchase behavior at that time, and *I* denotes the set of all items. After modeling the conditional probability of user purchasing items, the probability $a_{l,i}$ of a user purchasing item *i* after purchasing item *l* can be defined as

$$a_{l,i} = p(i \in B_t|l \in B_{t-1}). \tag{4.104}$$

This method allows for modeling the sequential dependencies between user purchase behaviors, which is helpful for capturing the transitional patterns in user preferences over time. It can be used to predict the likelihood of a user purchasing a particular item based on their previous purchase behavior.

In the paper by Steffen Rendle et al. [48], the Markov chain-based sequential modeling method was first introduced in the context of the Factorizing Personalized Markov Chains (FPMC) algorithm for recommendation systems. Based on the definition in the previous message, Steffen first defined the personalized conditional purchase probability for user *u* as follows:

$$a_{u,l,i} = p(i \in B_t^u|l \in B_{t-1}^u). \tag{4.105}$$

Here, $a_{u,l,i}$ represents the probability that user *u* purchases item *i* at time *t* given that they had previously purchased item *l* at time $t-1$. This personalized conditional probability is an essential component in the FPMC algorithm, which leverages the Markov chain-based modeling to capture the sequential patterns in user behaviors and make personalized recommendations accordingly.

Then, they proposed a factorization algorithm based on Tucker tensor decomposition:

$$\hat{\mathcal{A}} = \boldsymbol{C} \times_U \boldsymbol{V}^U \times_L \boldsymbol{V}^L \times_I \boldsymbol{V}^I. \tag{4.106}$$

Here, $\boldsymbol{C} \in \mathbb{R}^{k_U, k_L, k_I}$ is the core tensor, and $\boldsymbol{V}^U \in \mathbb{R}^{|U| \times k_U}$, $\boldsymbol{V}^L \in \mathbb{R}^{|L| \times k_L}$, and $\boldsymbol{V}^I \in \mathbb{R}^{|I| \times k_I}$ are the user feature matrix, last-purchased item feature matrix, and predicted item feature matrix, respectively. The parameter *k* represents the length of features and is a hyperparameter of the model. The algorithm uses the estimated transition probability tensor $\hat{\mathcal{A}}$ to predict the next item of interest for the user at the next time step:

$$\begin{aligned} \hat{p}\left(i \in B_t^u \mid l \in B_{t-1}^u\right) &= \frac{1}{|B_{t-1}^u|} \sum_{l \in B_{t-1}^u} \hat{a}_{u,l,t} \\ &= \frac{1}{|B_{t-1}^u|} \sum_{l \in B_{t-1}^u} \left(v_u^{U,I}, v_i^{I,U} + v_i^{I,L}, v_l^{L,I} + v_u^{U,L}, v_l^{L,U}\right). \end{aligned} \tag{4.107}$$

In this equation, $\hat{p}(i \in B_t^u \mid l \in B_{t-1}^u)$ represents the estimated probability that item *i* is in the user *u*'s next purchase sequence given that item *l* was in the user *u*'s previous purchase sequence. The formula involves the user and item feature vectors $\boldsymbol{V}^U$, $\boldsymbol{V}^L$, and $\boldsymbol{V}^I$, which are obtained through the Tucker tensor decomposition to model the sequential dependencies between user behaviors.

In FPMC (Factorizing Personalized Markov Chains), as mentioned earlier, the interest of the user in item *i* is independent of item *l*. Hence, the terms $v_u^{U,I}$ and $v_i^{I,U}$ can be moved outside the summation. This chapter was the first to apply the Markov chain model to personalized recommendation and proposed a personalized factorization for the transition probability matrix to reduce the complexity of estimating the transition probability matrix. However, FPMC algorithm faces challenges in addressing sparsity issues and may perform poorly in many long-tail distribution datasets, where specific user–item recommendations suffer from limited data and are difficult to accurately characterize by the model.

In another paper, He et al. introduced the Fossil algorithm [20], which uses item similarity modeling and high-order Markov chains for sequential recommendation. They made two improvements: First, they employed an item similarity matrix to model user preferences, replacing the global preference modeling for items, and utilized the historical behavior set $\mathcal{I}_u^+$ to

filter user preferences. Second, they introduced high-order Markov chains to consider the influence of the last L user actions on the current prediction. The part of the Fossil method that models the temporal user behavior is similar to FPMC, but it includes the weighting of high-order Markov chains for user preferences. In experimental comparisons, Fossil demonstrated better predictive performance and recommendation effectiveness compared to traditional Markov chain-based recommendation algorithm FMC and personalized Markov chain modeling method FPMC.

Deep Learning-Based Sequential Recommendation

The fourth category of methods is based on deep learning for sequential recommendation. Traditional sequential recommendation algorithms heavily rely on matrix factorization or factorization techniques to decompose high-dimensional and sparse user–item interaction matrices or Markov chain transition probability matrices into low-rank representations. Alternatively, some methods encode user interactions as one-hot encodings and use factorization machines for feature interaction modeling to predict user behavior and make sequential recommendations. While these methods to some extent support modeling and optimization of user behavior sequences, they face challenges in modeling longer user behavior sequences and handling complex feature interactions and representations.

As shown in Fig. 4.20, since 2015, with the widespread application of deep learning methods in recommendation systems and user feedback prediction tasks, a plethora of deep learning-based sequential recommendation algorithms have emerged, with a significant increase in related research publications in recent years. These deep learning-based approaches have shown promising potential to overcome the limitations of traditional methods and improve the performance of sequential recommendation systems.

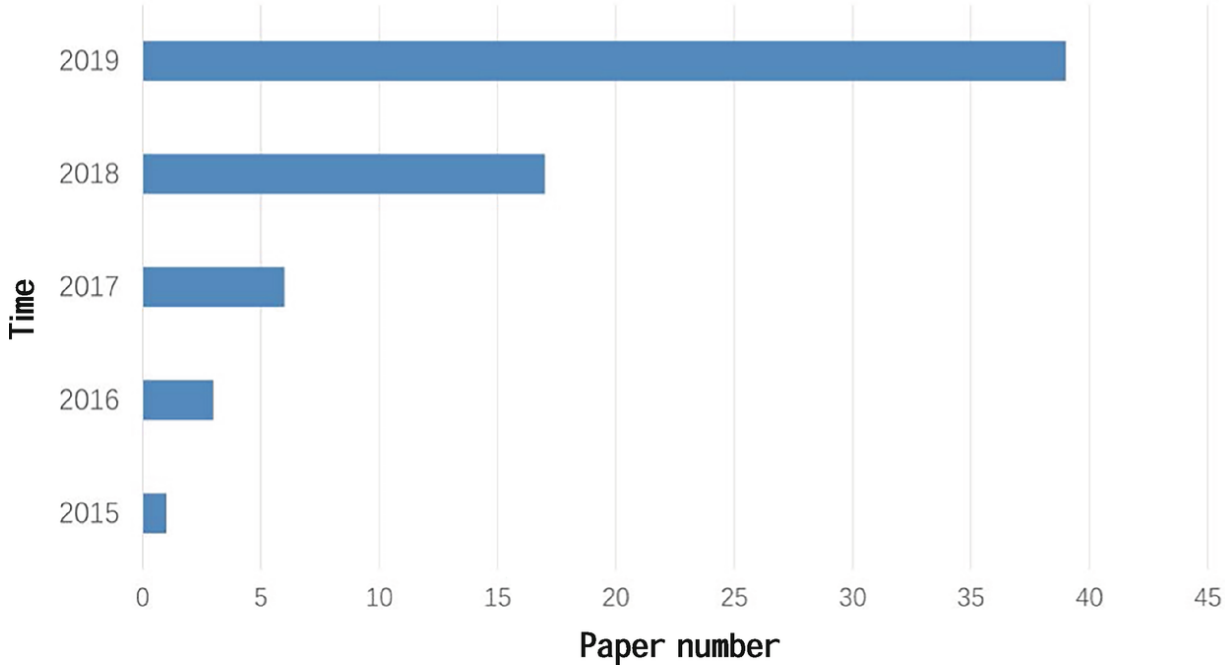


Fig. 4.20 The development trend of deep learning-based sequential recommendation algorithm papers [15]

Based on the perspective of deep learning models, deep learning-based sequential recommendation algorithms can be categorized into autoregressive recurrent neural networks, non-autoregressive deep learning models, and memory-based neural networks for modeling user behavior sequences. The introduction of deep learning methods aims to address challenges in modeling user behavior sequences, such as sequential pattern mining and modeling extremely long user behavior sequences. Some research works also explore joint modeling of user-side behavior sequences and item-side access user sequences to enhance the overall effectiveness of sequential recommendation. The following sections will provide a detailed overview and introduction of these latest algorithms.

4.4.3 Recurrent Neural Network-Based Sequential Recommendation

Hidasi et al. first introduced the use of a recurrent neural network (RNN) to model user behavior sequences in their paper [23], and the algorithm is named GRU4Rec. In this section, we will first introduce the structure details of the recurrent neural network unit used in GRU4Rec, followed by a detailed explanation of the modeling approach.

For information about recurrent neural networks, please refer to Sect. 3.3.2 of this book. The update equation of a standard recurrent neural network can be expressed as follows:

$$\boldsymbol{h}_t = g(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{h}_{t-1}). \tag{4.108}$$

Here, $\boldsymbol{h}_t$ represents the hidden state vector of the recurrent neural network at time step *t*, and it is updated based on the previous hidden state vector $\boldsymbol{h}_{t-1}$ and the current input vector $\boldsymbol{x}_t$ using a smooth and bounded function *g* as the activation function.

In the paper, Hidasi used the Gated Recurrent Unit (GRU), a type of recurrent neural network unit that mitigates the vanishing gradient problem commonly encountered in standard RNNs. The update equation of the GRU's hidden state is as follows:

$$\boldsymbol{h}_t = (1 - z_t)\boldsymbol{h}_{t-1} + z_t\hat{\boldsymbol{h}}_t. \tag{4.109}$$

The update gate value z_t is computed using the following formula:

$$z_t = \sigma(\boldsymbol{U}_z\boldsymbol{x}_t + \boldsymbol{W}_z\boldsymbol{h}_{t-1}). \tag{4.110}$$

Similarly, the candidate hidden state vector is computed as follows:

$$\hat{\boldsymbol{h}}_t = \tanh(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})) \tag{4.111}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{U}_r\boldsymbol{x}_t + \boldsymbol{W}_r\boldsymbol{h}_{t-1}). \tag{4.112}$$

In these equations, σ represents the sigmoid activation function, and $\odot$ denotes element-wise multiplication. The GRU allows for better capturing of long-term dependencies in the sequence data, making it more suitable for modeling user behavior sequences in the context of recommendation systems.

In the modeling process using recurrent neural networks, at the *t*-th time step, the paper authors use the items or item sequences that the user has previously interacted with in the same session as the input to the recurrent neural network unit, which first go through an embedding layer. If items are used as input, the input vector is directly passed through the embedding layer after one-hot encoding. If item sequences are used as input, the authors calculate a weighted sum of the embedding vectors of all items in the sequence (with higher weights for more recent interactions) before proceeding with the subsequent computations. Then, the information passes

through multiple gated recurrent units (GRUs) and a feedforward layer (a regular fully connected neural network layer) before finally outputting the current user's preference score for the item. The entire computation process is illustrated in Fig. 4.21. The algorithm's loss function consists of the Bayesian personalized ranking loss and the top-1 ranking loss function.

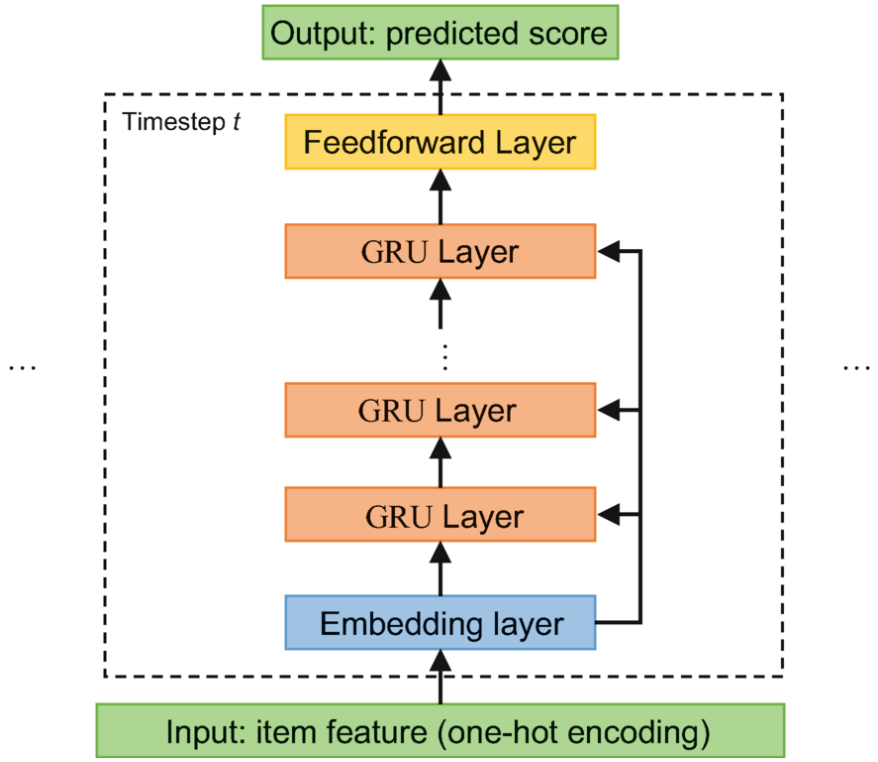


Fig. 4.21 Sequential recommendation based on recurrent neural networks [23]

GRU4Rec introduces several algorithmic innovations to achieve higher training efficiency. First, considering that different user sessions may have varying lengths, the authors process multiple session sequences in parallel as a batch of data and then use the recurrent neural network units to calculate in parallel from left to right. Additionally, at the end of the sequences in the batch data, a new session sequence is appended to support flexible computation requirements. Furthermore, considering the vast number of item categories in recommendation systems, it is computationally challenging to calculate the final scores for all items. Therefore, the authors adopt a weighted negative sampling strategy, where user's actual interacted items are used as positive examples for prediction and items from other sequences in the batch data are used as negative examples, creating a negative sampling scheme. Moreover, since different batches of data contain all items that users have ever interacted with, this negative sampling strategy is also based on

popularity statistics, meaning that items that have been interacted with by more users are more likely to be sampled.

GRU4Rec, as a classic recurrent neural network algorithm for sequence recommendation, has established the fundamental operational logic of using recurrent neural networks for sequence recommendation. Subsequent methods mostly follow the same construction approach, so the following content will not redundantly elaborate on similar algorithm details. The advantage of this algorithm lies in its use of autoregressive neural network structure to model the non-linear variations in user interests and behavior patterns. It also considers the impact of both the user's past historical behavior and the most recent behavior on their interests in the next time step. However, there are also some drawbacks to such methods.

These methods have high computational complexity and slow computation speed. Modeling user historical behavior sequences involves sequentially inputting and computing from older to the most recent behaviors in a recurrent neural network unit. Each unit's computation logic is relatively complex. If the sequence length is long, the computation time increases linearly, leading to significant overall time consumption. This high computational cost may not be feasible for real-time online systems.

As a result, several solutions have been proposed to address these challenges.

4.4.4 Non-autoregressive Neural Network-Based Sequence Modeling

Non-autoregressive neural network-based sequence modeling mainly includes two types of models: Convolutional Neural Networks (CNNs) and transformer models. This section will introduce the classic algorithms for both types of methods.

Convolutional Neural Networks were originally applied in computer vision and image processing. In the context of sequence recommendation, the user's accessed item feature sequences are arranged in chronological order, forming an "image-like" structure. The idea behind using Convolutional Neural Networks for modeling user behavior sequences is similar to processing image data. Tang et al. proposed the Caser (Convolutional Sequence Embedding Recommendation) algorithm in their paper [55]. It consists of three main modules: the embedding lookup module,

the convolutional layer module, and the fully connected layer module. The specific computation flow of each module will be introduced below.

The embedding lookup module, as shown on the left side of Fig. 4.22, in the Caser algorithm, combines the one-hot encoded representations of the user u's recent L historical behavior sequences at time t into a large tensor. This tensor is then interacted with the embedding dictionary $\boldsymbol{M}$ to obtain a new representation composed of embedding vectors, denoted as $\boldsymbol{E}^{(u,t)} = [\boldsymbol{Q}_{S_{t-L}^u}, \cdots, \boldsymbol{Q}_{S_{t-2}^u}, \boldsymbol{Q}_{S_{t-1}^u}]^\top \in \mathbb{R}^{L \times d}$, where $\boldsymbol{Q}_{S_t^u} \in \mathbb{R}^d$ is the *d*-dimensional real-valued embedding vector representing the item that user u interacts with at time t. At the same time, each user also obtains their own user embedding vector $\boldsymbol{P}_u \in \mathbb{R}^d$ for subsequent modeling and computations.

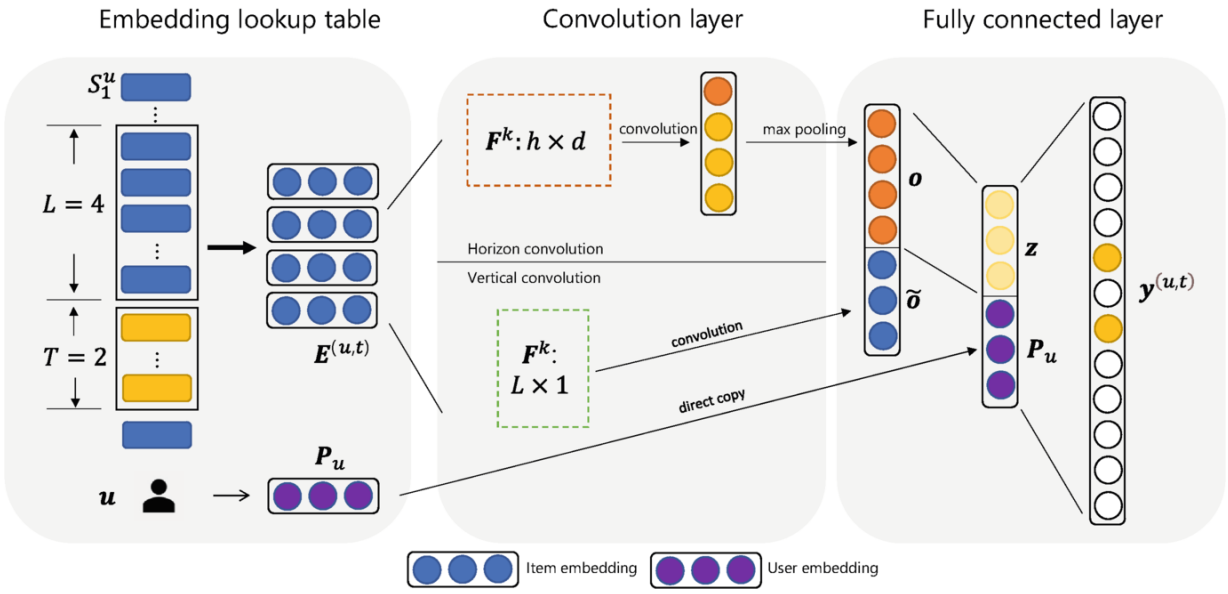


Fig. 4.22 User behavior sequence modeling based on Convolutional Neural Networks [55]

The convolutional layer module applies convolutional operations to the feature tensor obtained earlier. First, the Caser algorithm considers the $L \times d$-dimensional real-valued matrix $\boldsymbol{E}$ as an "image" composed of the representation vectors of the previous *L* user historical behaviors in the latent space. Next, as shown in the middle part of Fig. 4.22, the Caser algorithm uses two types of operations: horizontal convolution and vertical convolution, to apply convolutional operations to the user representation tensor. The horizontal convolution kernel is an $h \times d$-dimensional matrix, and in the experiments, the authors set *n* horizontal convolution kernels

$\boldsymbol{F}^k, 1 \leq k \leq n$, each with a parameter $h = 2$, meaning that each convolution operation constructs a modeling window of length 2. The convolution operations are performed by scanning the matrix $\boldsymbol{E}$ row by row from top to bottom. Additionally, the algorithm also uses vertical convolution operations with $\tilde{n}$ $L \times 1$-dimensional convolution kernels $\tilde{\boldsymbol{F}}^k, 1 \leq k \leq \tilde{n}$, scanning the matrix $\boldsymbol{E}$ column by column from left to right. Through these operations, the Caser algorithm can discover and model complex co-occurrence relationships among item sequences, leading to better predictions for user interests and potential future interactions with items. The calculation process of the convolution operation can be mathematically expressed as follows:

$$\boldsymbol{c}_i^k = \phi_c(\boldsymbol{E}_{i:i+h-1} \odot \boldsymbol{F}^k), \ 1 \leq i \leq L - h + 1 \tag{4.113}$$

$$\boldsymbol{c}^k = [c_1^k, c_2^k, \cdots, c_{L-h+1}^k] \tag{4.114}$$

$$\tilde{\boldsymbol{c}}^k = [\tilde{\boldsymbol{c}}_1^k, \tilde{\boldsymbol{c}}_2^k, \cdots, \tilde{\boldsymbol{c}}_d^k] \tag{4.115}$$

$$\tilde{\boldsymbol{c}}^k = \sum_{l=1}^{L} \tilde{\boldsymbol{F}}_l^k \cdot \boldsymbol{E}_l. \tag{4.116}$$

In these equations, ϕ_c is the activation function of the convolutional layer; $\odot$ denotes the element-wise Hadamard product (matrix-wise multiplication); $\boldsymbol{E}_l$ represents the *l*-th row of the original behavior sequence embedding matrix $\boldsymbol{E}$; *k* is the index of different convolution kernels $\boldsymbol{F}$; $\boldsymbol{F}$ represents the horizontal convolution kernel, and $\tilde{\boldsymbol{F}}$ represents the vertical convolution kernel. Additionally, after the horizontal convolution layer computation, the results will undergo a max-pooling layer, denoted as $\boldsymbol{o} = \{\max(\boldsymbol{c}^1), \max(\boldsymbol{c}^2), \cdots, \max(\boldsymbol{c}^n)\}$, where the max-pooling operation selects the maximum value from each vector $\boldsymbol{c}^k$ obtained by different convolution kernels $\boldsymbol{F}$ and discards the other values, resulting in an *n*-dimensional vector $\boldsymbol{o} \in \mathbb{R}^n$ used for subsequent computations. On the other hand, the results obtained after the vertical convolution operation do not go through the max-pooling layer; they are directly combined into a tensor $\tilde{\boldsymbol{o}} = [\tilde{\boldsymbol{c}}^1, \tilde{\boldsymbol{c}}^2, \cdots, \tilde{\boldsymbol{c}}^{\tilde{n}}] \in \mathbb{R}^{d\tilde{n}}$.

The last module is the fully connected layer, as shown on the right side of Fig. 4.22. Here, a single-layer fully connected neural network is taken as

an example, and the specific computation is given by

$$\boldsymbol{z} = \phi_a\left(\boldsymbol{W}\begin{bmatrix}\boldsymbol{o}\\ \tilde{\boldsymbol{o}}\end{bmatrix} + \boldsymbol{b}\right), \tag{4.117}$$

where $\boldsymbol{W} \in \mathbb{R}^{d\times(n+d\tilde{n})}$. The final prediction network considers the output of the previous user behavior sequence modeling and the user embedding representation vector as inputs to predict the final click probability:

$$\boldsymbol{y}^{(u,t)} = \boldsymbol{W}'\begin{bmatrix}z\\ \boldsymbol{P}_u\end{bmatrix} + \boldsymbol{b}', \tag{4.118}$$

where the prediction vector $\boldsymbol{y}^{(u,t)}$ represents the probability of user *u* interacting with different items at time *t*.

Currently, the sequence modeling methods introduced in this chapter are all based on the same assumption that user interests are related to recent interactions. However, these methods mainly focus on short-term user modeling, limited by the user's most recent behavior. Nowadays, users have accumulated very long behavior sequences on different online platforms, as shown in Table 4.1. If we consider longer user behavior sequences, existing methods may not effectively model periodic, diverse, and continuously evolving user interests. Zhang et al. [80] proposed an additional static user representation vector to model the inherent user interests, which are not influenced by time. However, such methods overlook the diverse characteristics of user interests. Ying et al. [77] proposed a multi-layer attention approach based on user behavior sequence features to model long-term user interests. However, their model can only capture relatively simple behavioral patterns and does not consider long-term and multi-scale behavioral dependencies. Furthermore, existing research has almost ignored modeling the user's lifelong behavior sequence, thus failing to provide a complete and comprehensive user profile.

4.4.5 Self-attention-Based Sequence Recommendation

In addition to Caser, another type of non-autoregressive sequence modeling algorithm used for sequential recommendation is based on the self-attention mechanism. This class of methods was initially proposed by Kang et al. [29], and later, other related research work was published [52]. The modeling approach of these methods is similar to GRU4Rec and Caser, but they use different models for modeling and computing user behavior sequences.

Unlike other non-autoregressive sequence modeling algorithms, the SASRec algorithm proposed by Kang et al. utilizes the transformer model [56]. The calculation process of the transformer model is different from the convolutional neural network, as it employs the self-attention mechanism to model user historical behavior sequences. Moreover, in the process of using self-attention, Kang et al. introduced causality modeling, which means that when modeling the sequence at time *t*, the information after time *t* is hidden to prevent information leakage. In the following sections, this chapter will introduce the self-attention-based sequence recommendation model, SASRec.

The SASRec model, like other sequence recommendation models, starts with an embedding representation layer. The original behavior sequence $(S_1^u, \cdots, S_L^u)$ of user *u* is transformed using the item embedding dictionary $\boldsymbol{M}$ to obtain a representation matrix $\boldsymbol{E} \in \mathbb{R}^{L \times d}$ with length *L*. Additionally, the relative positions of items in the sequence are modeled using position embeddings $\boldsymbol{P}$, which are then added to the original behavior sequence embedding matrix to obtain the sequence input matrix:

$$\widehat{\boldsymbol{E}} = \begin{bmatrix} \boldsymbol{E}_{s_1} + \boldsymbol{P}_1 \\ \boldsymbol{E}_{s_2} + \boldsymbol{P}_2 \\ \vdots \\ \boldsymbol{E}_{s_L} + \boldsymbol{P}_L. \end{bmatrix} \tag{4.119}$$

Here, $\boldsymbol{P}_l$, where $l \in [1, L]$, represents the position embedding vector of the *l*-th user behavior representation. Next, the input representation matrix $\widehat{\boldsymbol{E}}$ will be passed through the self-attention layer for further computation.

In Sect. 3.3.3 of this book, the computation of the attention mechanism has been introduced. Let us review the calculation method of the attention module:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}. \tag{4.120}$$

Here, $\boldsymbol{Q}$ represents the query matrix, $\boldsymbol{K}$ represents the key matrix, and $\boldsymbol{V}$ represents the value matrix (where each row represents an item). The dot product operation interacts the corresponding rows of the query matrix with the key matrix and calculates the weighted values for each row of the value

matrix. In natural language processing, the key matrix and value matrix are usually set to be the same, that is, $\boldsymbol{K} = \boldsymbol{V}$.

The specific computation process of the representation matrix $\widehat{\boldsymbol{E}}$ through the self-attention layer is as follows:

$$\boldsymbol{X} = \mathrm{SA}(\widehat{\boldsymbol{E}}) = \mathrm{Attention}(\widehat{\boldsymbol{E}}\boldsymbol{W}^{Q}, \widehat{\boldsymbol{E}}\boldsymbol{W}^{K}, \widehat{\boldsymbol{E}}\boldsymbol{W}^{V}). \tag{4.121}$$

The representation matrix is first passed through three separate linear mappings, ensuring consistent mapping results. Then, the query, key, and value matrices are input into the attention calculation formula successively, yielding the comprehensive representation matrix $\boldsymbol{X}$.

One important point to note here is that the modeling and prediction of user behavior sequences have "temporality". When predicting the user behavior at time step $(l+1)$, we cannot have knowledge of the user behavior after the prediction time, i.e., $S^{u}_{(l+1):L}$. Therefore, when performing the self-attention layer calculation for the sequences, it is necessary to introduce causality modeling, which means that the computation result at position *i* must be dependent on the result at position *j* where $(j < i)$. Thus, when implementing the attention calculation, it is crucial to ensure that all computations involving $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ maintain temporal dependency, in order to avoid information leakage.

Considering that all operations, including attention calculations, performed so far are linear, in order to introduce non-linearity into the model, a non-linear layer is added after the self-attention layer:

$$\boldsymbol{F}_i = \mathrm{FFN}(\boldsymbol{X}_i) = \mathrm{ReLU}\left(\boldsymbol{X}_i\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\right)\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}. \tag{4.122}$$

Here, $\boldsymbol{W}^{(\cdot)} \in \mathbb{R}^{d\times d}$ are square matrices, and $\boldsymbol{b}^{(\cdot)} \in \mathbb{R}^{d}$ are vectors.

It is important to note that $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ do not interact in this operation, and the entire process maintains the sequence dependency.

After introducing the computation of the attention layer, the SASRec model proceeds to stack multiple layers of self-attention for enhancing the modeling and expressive capacity of the model:

$$\boldsymbol{X}^{(b)} = \mathrm{SA}\left(\boldsymbol{F}^{(b-1)}\right), \tag{4.123}$$

$$\boldsymbol{F}_i^{(b)} = \mathrm{FFN}\left(\boldsymbol{X}_i^{(b)}\right), \forall i \in \{1, 2, \cdots, L\}. \tag{4.124}$$

For deep neural networks, stacking too many layers or a very deep network may lead to training instability and even overfitting issues. To address this, SASRec introduces three mechanisms: residual connection, layer normalization, and dropout. The residual connection is mainly inspired by the implementation of deep residual networks by He et al. [29]. The layer normalization formula is given by

$$\mathrm{LayerNorm}(\boldsymbol{x}) = \boldsymbol{\alpha} \odot \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta. \tag{4.125}$$

For any input vector $\boldsymbol{x}$, the layer normalization operation first computes the mean μ and standard deviation σ and then performs normalization. It is followed by an element-wise multiplication with the learnable parameter $\boldsymbol{\alpha}$, and finally, the bias term β is added. The dropout operation was initially introduced by Srivastava et al. [51]. It involves two steps: during training, some neural units are randomly set to zero with a probability value *p* to deactivate them, and during inference, the dropout operation is turned off to keep all neural units activated.

In the final prediction stage, the SASRec model performs an interaction operation between the output of the last layer and the entries of the item embedding dictionary:

$$r_{u,v,t} = \left(\boldsymbol{U}_u + \boldsymbol{X}_t^{(b)}\right) \boldsymbol{M}_v^{\top}. \tag{4.126}$$

This calculates the preference probability $r_{u,v,t}$ of user *u* toward item *v* at time step *t*. Here, $\boldsymbol{U}_u$ represents the user embedding vector for user *u*.

The training labels of the model consist of the user's real behavior sequence $(S_1^u, \cdots, S_{|S^u|-1}^u)$, which is defined as

$$o_t = \begin{cases} S_{|S^u|}^u, & t = L \\ S_{t+1}, & 1 \le t < L \\ \langle \mathrm{pad} \rangle, & \mathrm{otherwise} \end{cases}. \tag{4.127}$$

The real label at different time steps is the user's real behavior at the next time step, and $\langle \mathrm{pad} \rangle$ is used to fill the parts of the user behavior sequence that are less than *L* and has no real meaning. Finally, the SASRec model is trained and optimized using binary cross-entropy loss function.

4.4.6 Memory-Based Neural Networks for Sequential Recommendation

In order to model longer user behavior sequences more effectively, many researchers have turned their attention to memory-based neural networks. The idea of memory-based neural networks for sequence modeling was initially proposed by Chen et al. in a research paper [9]. The authors drew inspiration from the design of external memory networks in natural language processing [16, 69]. They maintained a private memory matrix $\boldsymbol{M}^u$ for each user, and when predicting the interaction probability of user *u* with item *i*, they used the item representation $\boldsymbol{q}_i$ for memory querying, obtaining the memory representation vector:

$$\boldsymbol{p}_u^m = \mathrm{READ}(\boldsymbol{M}^u, \boldsymbol{q}_i). \tag{4.128}$$

This memory-based approach allows the model to retrieve relevant information from the user's private memory when making predictions for user–item interactions.

Next, the algorithm concatenates the memory representation vector $\boldsymbol{p}_u^m$ with the user's intrinsic representation vector $\boldsymbol{p}_u^*$, where the concatenation function is implemented as a weighted sum in the research paper:

$$\mathrm{MERGE}(x, y) = x + \alpha y \tag{4.129}$$

$$\boldsymbol{p}_u = \mathrm{MERGE}(\boldsymbol{p}_u^*, \boldsymbol{p}_u^m) = \boldsymbol{p}_u^* + \alpha \boldsymbol{p}_u^m. \tag{4.130}$$

Here, α represents the weight coefficient, which is determined by the algorithm's training personnel. When predicting the user–item interaction probability $\hat{y}_{ui}$, the authors of the paper used a prediction function similar to matrix factorization:

$$\hat{y}_{ui} = \sigma\left(\boldsymbol{p}_u^\top \cdot \boldsymbol{q}_i\right). \tag{4.131}$$

In this function, σ represents the sigmoid activation function, and the dot product $\boldsymbol{p}_u^\top \cdot \boldsymbol{q}_i$ calculates the interaction score between the user and the item.

Regarding the user's private memory module, there is an update function that operates on the memory matrix after the user interacts with a specific item. The update function is defined as follows:

$$\boldsymbol{M}^u = \mathrm{WRITE}(\boldsymbol{M}^u, \boldsymbol{q}_i). \tag{4.132}$$

As shown in Fig. 4.23, the user representation vector is used for both reading (READ) and writing (WRITE) operations on the user's memory module.

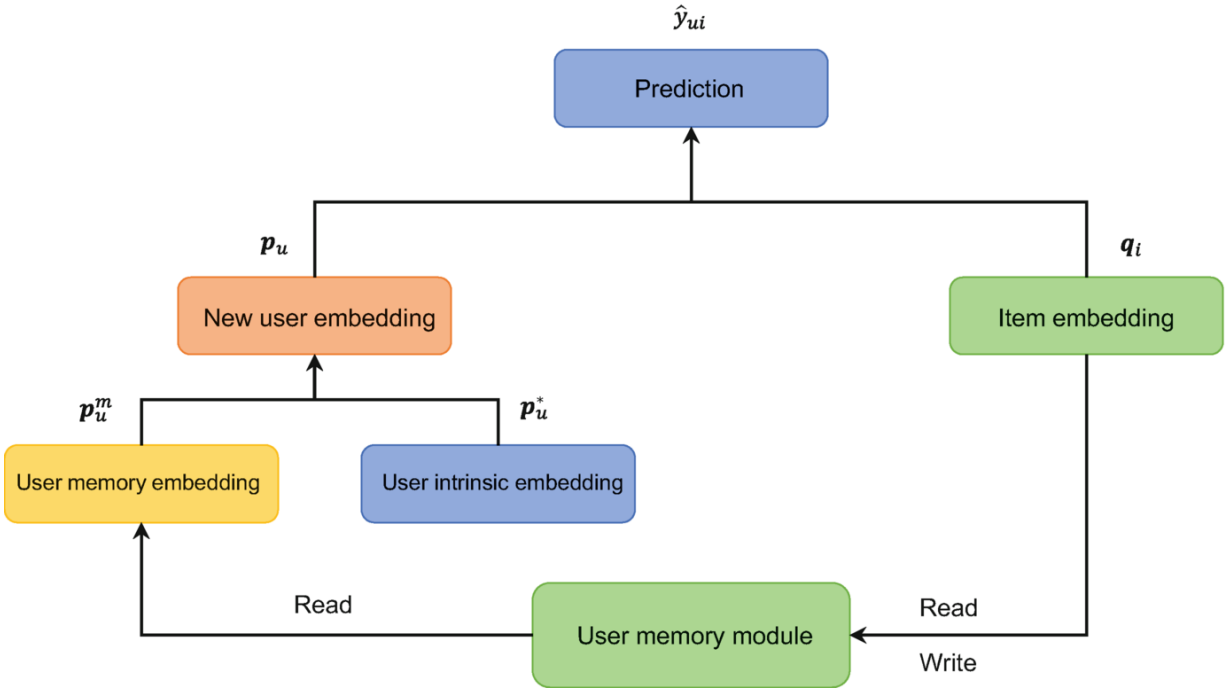


Fig. 4.23 The framework of the sequence recommendation model RUM based on memory network [9]

In their RUM paper [9], Chen et al. implemented two types of read–write schemes, and this chapter will introduce one of them: the item-level memory module. This memory module stores a set of items that the user recently interacted with, denoted as $I_u^+ = \{v_1^u, v_2^u, \cdots, v_{|I_u^+|}^u\}$, where v_i^u represents the i-th item visited by user u. In the entire system, the user representation vector $\boldsymbol{p}_u^* \in \mathbb{R}^d$ and the item representation vector $\boldsymbol{q}_{v_i^u} \in \mathbb{R}^d$ are d-dimensional real-valued vectors. The user memory module $\boldsymbol{M}^u \in \mathbb{R}^{d \times K} = \{\boldsymbol{m}_1^u, \boldsymbol{m}_2^u, \cdots, \boldsymbol{m}_K^u\}$ consists of K memory slots used to store user-related memory representation vectors.

For the READ operation of the memory module, Chen employed an attention-based memory read mechanism in the paper, calculating the relevance between the current item $\boldsymbol{q}_i$ and the contents of the memory module $\boldsymbol{M}^u$:

$$w_{ik} = \boldsymbol{q}_i^\top \cdot \boldsymbol{m}_k^u, z_{ik} = \frac{\exp(\beta w_{ik})}{\sum_j \exp(\beta w_{ij})}, \ \forall k = 1, 2, \cdots, K. \tag{4.133}$$

Then, RUM utilizes the computed weights to read the memory contents for subsequent prediction:

$$\boldsymbol{p}_u^m = \sum_{k=1}^{K} z_{ik} \cdot \boldsymbol{m}_k^u. \tag{4.134}$$

For the WRITE operation of the memory module, only the items that the user has actually interacted with $\boldsymbol{q}_{v_i^u} \in I_u^+$ will affect the contents of the memory module, namely $\boldsymbol{M}^u = \left\{\boldsymbol{q}_{v_{i-1}^u}, \boldsymbol{q}_{v_{i-2}^u}, \cdots, \boldsymbol{q}_{v_{i-K}^u}\right\}$. When a new item that the user interacts with is added to the memory module, its contents will be updated to $\boldsymbol{M}^u = \left\{\boldsymbol{q}_{v_i^u}, \boldsymbol{q}_{v_{i-1}^u}, \cdots, \boldsymbol{q}_{v_{i-K+1}^u}\right\}$, where $\boldsymbol{q}_{v_i^u}$ enters the memory module and replaces the entry of the oldest user-interacted item. The other feature-based memory module read–write mode can be referred to in the original paper, and the overall approach is similar to the above scheme, which will not be reiterated here.

The sequence recommendation algorithm based on memory neural networks reduces the complexity of modeling long sequences in sequence recommendation. By utilizing the external memory network module's space, it decreases the linear complexity of modeling long sequences during each user–item interaction probability prediction, greatly improving prediction efficiency. Moreover, for modeling interactions with items from the more distant past, larger memory networks can be used. However, the memory capacity of the memory module in memory neural network-based methods is still limited, and it has not yet solved the problem of modeling extremely long user behavior sequences or even lifelong user behavior sequences. As user behavior sequences become increasingly rich, and the proportion of existing users increases, Internet platforms are facing algorithmic challenges in modeling extremely long sequences.

Ren et al. made improvements based on memory networks and proposed a solution called HPMN (Hierarchical Periodic Memory Network) for modeling lifelong user behavior sequences. The authors divided the memory module into *L* layers and used different update frequencies for each layer. For the *l*-th layer memory module, its update frequency is 2^{l-1}. For example, if the user memory module is divided into three layers, the update frequency for the first layer is 1, which means the content of this layer's memory module will be updated every time the user interacts with an item. The update frequencies for the second and third layers are $2^{2-1} = 2$ and

$2^{3-1} = 4$, respectively, which means the contents of these two layers will be updated after the user interacts with 2 and 4 items, respectively. Additionally, when updating the memory content in each layer, the next layer's memory content is utilized as well, specifically:

$$\boldsymbol{m}_{l,i}^{u} = \mathrm{WRITE}\left(\boldsymbol{q}_{i}^{u}, \boldsymbol{m}_{l,i-1}^{u}, \boldsymbol{m}_{l-1,i}^{u}\right), \quad l = 1, 2, \cdots, L \tag{4.135}$$

$$\boldsymbol{m}_{0,i}^{u} = \mathrm{WRITE}\left(\boldsymbol{q}_{i}^{u}, \boldsymbol{m}_{0,i-1}^{u}\right). \tag{4.136}$$

Here, $\boldsymbol{m}_{l,i}^{u}$ represents the memory content of the *l*-th layer of the current user memory network, $\boldsymbol{m}_{l,i-1}^{u}$ is the memory content from the previous update, and $\boldsymbol{m}_{l-1,i}^{u}$ is the memory content from the next layer. This hierarchical memory network update method fully utilizes the memory contents between layers, passing fine-grained memory contents to coarser-grained memory layers. Moreover, the memory contents of different layers are updated using the latest memory content from the previous update, achieving a similar effect to that of recurrent neural networks.

When making predictions, the HPMN hierarchical periodic memory network's modeling results were visualized, revealing interesting findings. As shown in Fig. 4.24, the top-down visualization displays the modeling results for three users. The left-side vector $\boldsymbol{w} = [\cdots, w^{l}, \cdots] \in \mathbb{R}^{6}$ shows the weighted coefficients of different layers' memory contents after the memory network READ operation, where the index *l* indicates the importance of the memory contents from layers with larger indices and corresponds to coarser-grained levels, suggesting higher relevance for predictions.

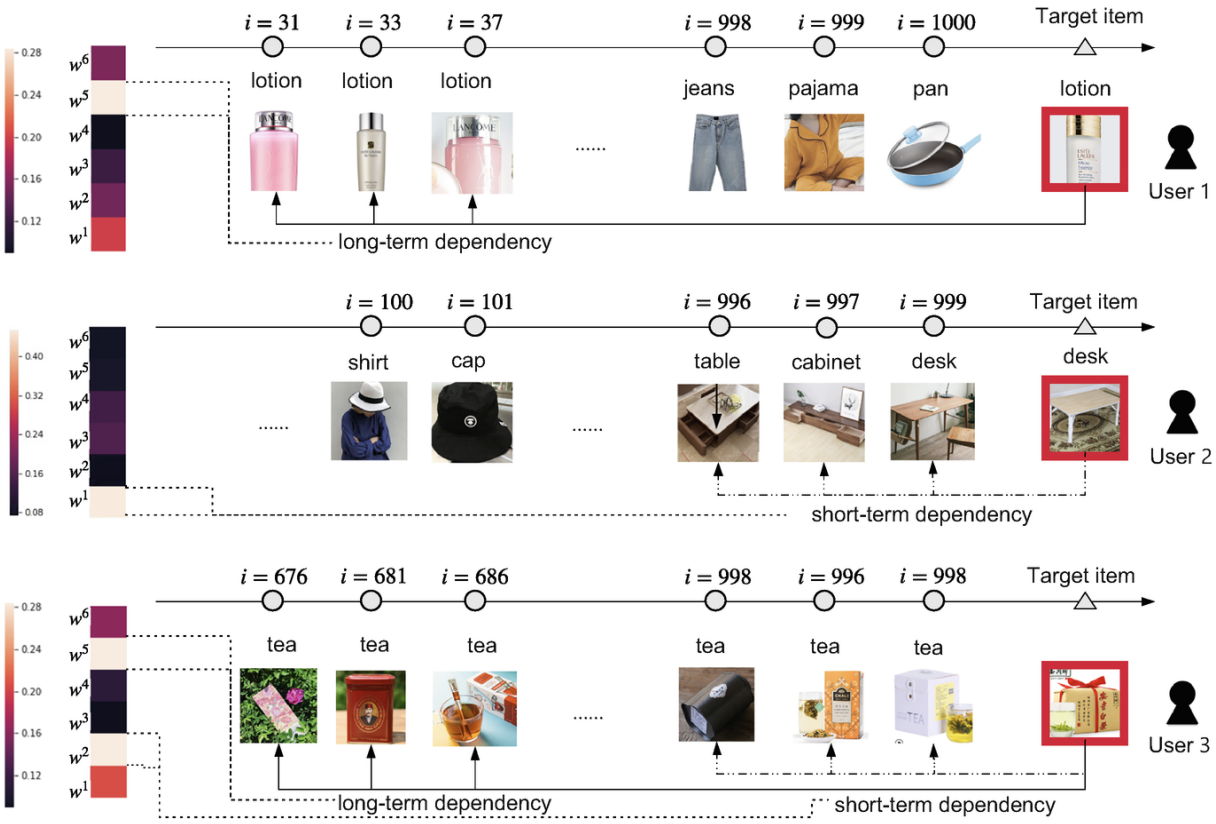


Fig. 4.24 Visualization of the modeling results of the hierarchical periodic memory network (HPMN) [9]

From the visualization in the figure, we can observe interesting patterns for three users' modeling results using HPMN [9]. For User 1, who is currently interested in buying lotion, the historical behavior sequence shows a strong correlation with makeup products related to lotion. As a result, the coarser-grained hierarchical memory network receives higher weights for its content. On the other hand, for User 2, whose interest in the item "desk" is more related to recent behaviors, the memory network's weights are higher on the finer-grained hierarchical levels. As for User 3, whose behavior sequence mainly consists of "tea"-related items, HPMN balances the utilization of coarse- and fine-grained memory levels in modeling due to the consistent interest pattern.

These three examples vividly demonstrate that HPMN can capture various temporal patterns of user behaviors, effectively modeling and optimizing for long user behavior sequences.

With this hierarchical memory module design, the HPMN model can utilize different update frequencies to model user access item sequences with different spans at different granularities. It captures both long-span coarse-

grained temporal patterns and short-span fine-grained temporal patterns, which greatly benefits user behavior modeling and recommendation. The memory layers with longer spans can be used to remember stable interest patterns of users, while the layers with shorter spans are utilized to model users' frequent and dynamic changes in interests and behaviors.

4.4.7 User–Item Dual Sequence Modeling

In the traditional recommender systems, researchers have generally given similar attention to both users and items. Algorithms such as matrix factorization and factorization machines represent users and items as latent space vectors. They model users' interests in items through interactions and predict the probability of interactions (such as click-through rate or purchase conversion rate). The previous sections have focused on modeling user behavior sequences, where these algorithms only pay attention to the sequences of items that users interact with and model these sequences accordingly. However, the attention to the item side is usually limited to obtaining latent vectors to represent the items. This raises a natural question: should we also pay attention to the user access sequences on the item side?

Wu et al. proposed a comprehensive modeling approach that utilizes both user-side and item-side sequences in their paper [70], known as the recurrent recommender network (RRN). As shown in Fig. 4.25, for a user *u*, their historical interactions with several items before the current time form the user's historical behavior sequence. On the other hand, the users who have accessed a particular item *v* form the item-side user sequence. Both sequences are individually modeled using recurrent neural networks (RNNs). The learned hidden vectors and user representations, along with item representations, are then fed into the prediction function to predict the final interaction probability r_{uv}.

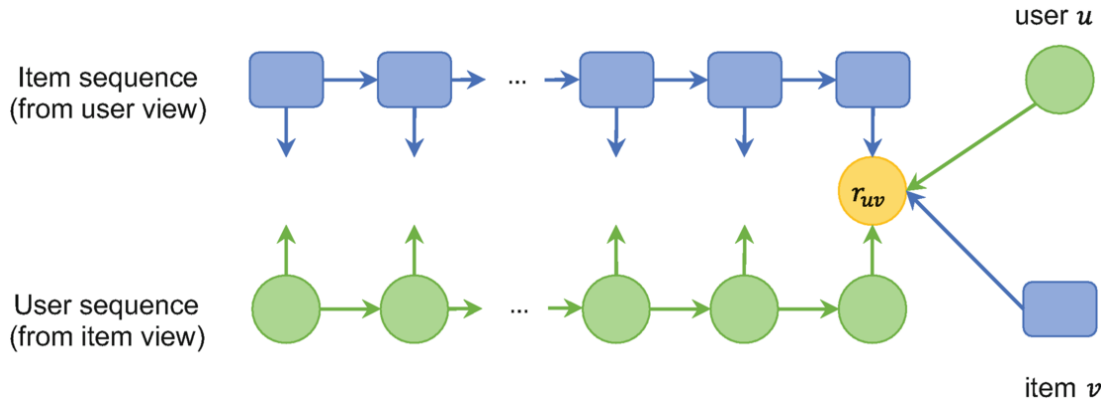


Fig. 4.25 Recurrent recommender network [70] illustration

Other algorithms [45] have also adopted similar ideas. The assumption behind these algorithms is that the item-side user access sequences represent the popularity of items and their influence patterns. For example, many items have seasonal characteristics, such as New Year's cards being more popular during the New Year holiday but not as much during other time of the year. By using item-side user access sequences, the model can understand which items are currently popular among users. By combining this with user-side pattern mining and matching, the recommendation algorithm can better recommend items to the corresponding users. Similarly, the algorithm can also recommend suitable users for specific items.

In summary, with the widespread use and dissemination of Internet products, these online platforms are facing a massive demand to serve existing users. Meanwhile, users are generating a large number of historical behaviors, leading to the emergence of sequential recommendation systems and algorithms. From an algorithmic perspective, sequence recommendation can be categorized into four types: sequence pattern mining methods, time matrix factorization methods, Markov chain-based modeling methods, and deep learning methods. This section introduced several representative algorithms under these categories and provided a detailed explanation of several deep learning models based on the length of user sequences. Through these introductions, the aim is to give readers an overall impression of the current state of sequential recommendation algorithms and help them understand and keep track of the cutting-edge algorithms in this field.

4.5 Recommender Systems Combined with Knowledge Graph

In 2012, Google proposed the concept of “Knowledge Graph”, which was originally intended to show the results returned by search engines in a more intelligent form to optimize the user experience. Nowadays, knowledge graph has been widely used in many application fields, such as search, question answering, recommendation, text understanding, and generation. Knowledge graph often uses structured triplets to describe the relations between entities. For example, <Bill Gates, founds, Microsoft> is a knowledge triplet. “Bill Gates” and “Microsoft” are two entities, represented as two nodes in the knowledge graph, and “founds” is the relation between them, represented as the edge connecting the two nodes. The relations between nodes and edges can be a many-to-many relation. Because Microsoft was co-founded by Bill Gates and Paul Allen, there is another triplet <Paul Allen, founds, Microsoft> in the knowledge graph. The knowledge graph establishes the relationship between entities with explicit semantic relations, so it can directly bring three benefits to the recommender system. The first is to enrich the user–item and item–item connections, to reduce the sparsity of user behavior. In the recommender system, the interaction between users and items is often very sparse. Through the knowledge graph, different levels of entity information can be used to get the hidden relations between items, so as to establish more relations between users and items. As shown in Fig. 4.26, if a user has seen the movie “Back to the Future”, it is likely that the user will also like the movie “Forest Gump”, since they are both directed by Robert Zemeckis. The second is to enrich the attributes of items, so as to learn a more comprehensive item representation and improve the accuracy of recommendation. The third is to use the semantic relations in the knowledge graph to provide explainability for the recommender system. For example, in Fig. 4.26, when recommending the movie “Forrest Gump” to users, several recommendation reasons can be generated based on the knowledge graph, including “The Green Mile – Tom Hanks – Forrest Gump” and “Back to The Future – Robert Zemeckis – Forrest Gump”. This section will focus on these three aspects to introduce the recommendation algorithms combined with knowledge graph.

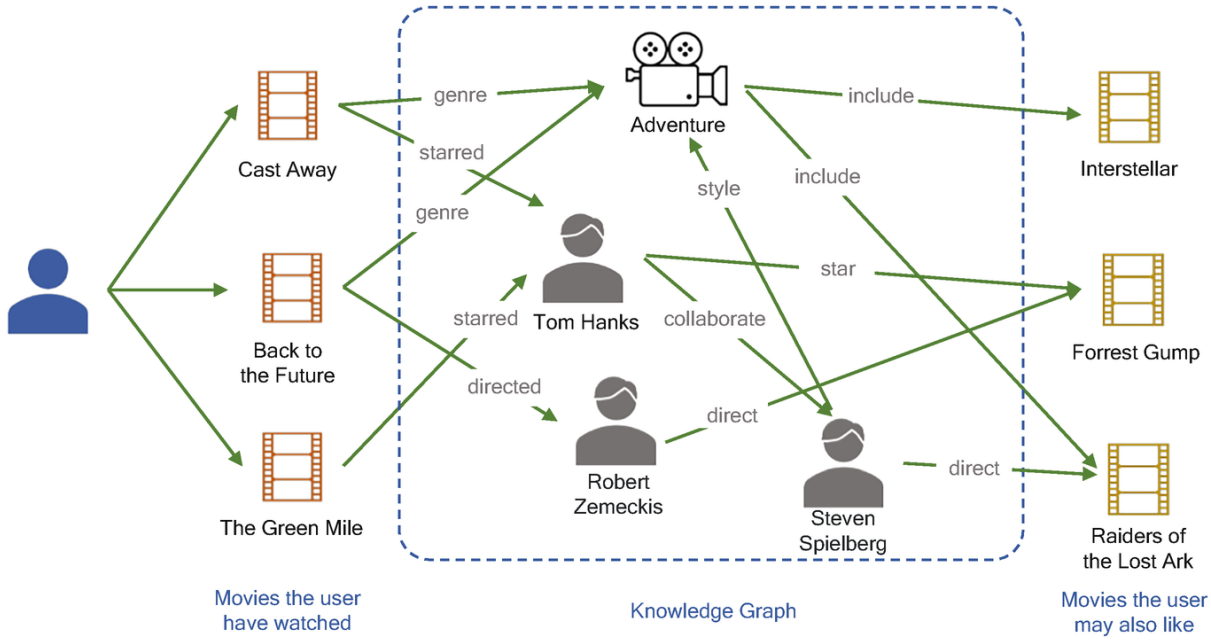


Fig. 4.26 Example of Knowledge Graph Enhanced Movie Recommendation

4.5.1 Enhancing User–Item Interaction Modeling

The RippleNet Model

In the traditional item-based collaborative filtering, the user *u* is represented by a collection of items $C(u)$ that he/she has interacted with, and the item *v* to be predicted directly interacts with $C(u)$ to get the predicted value. Knowledge graph brings rich relations among items. An intuitive use is to expand the set of items that interest the user, so that the item *v* to be predicted can interact with more content and reduce data sparsity. Based on this idea, Hongwei Wang et al. [60] proposed RippleNet. By simulating the ripples propagating on the water, RippleNet spreads and diffuses the user's interests on the knowledge graph, narrowing the distance between users and unknown items. Let $\mathcal{G} = \{(h, r, t)|\mathcal{E}, \mathcal{R}\}$ denote the knowledge graph, $h, t \in \mathcal{E}$ denote the head entity and the tail entity in a triplet, respectively, and $r \in \mathcal{R}$ represent the relation between the two entities. It is assumed that items in the recommender system can correspond to entities in the knowledge graph. In order to facilitate the representation of users' extended interests in the knowledge graph, two terms are defined: relevant entities and ripple set.

Definition 1 (RELEVANT ENTITIES) Given the user–item interaction matrix $\boldsymbol{Y}$ and the knowledge graph $\mathcal{G}$, the set of *k*-hop relevant entities for

user *u* is

$$\mathcal{E}_u^k = \{t|(h,r,t) \in \mathcal{G}, h \in \mathcal{E}_u^{k-1}\}, k = 1,2,\cdots,H, \tag{4.137}$$

where $\mathcal{E}_u^0 = C(u) = \{v|y_{uv} = 1\}$ is the set of items that the user has interacted with.

Definition 2 (RIPPLE SET) The *k*-hop ripple set of user *u* is defined as the set of the knowledge graph triplets whose head entities belong to $\mathcal{E}_u^{k-1}$

$$\mathcal{S}_u^k = \{(h,r,t)|(h,r,t) \in \mathcal{G}, \text{and } h \in \mathcal{E}_u^{k-1}\}, k = 1,2,\cdots,H. \tag{4.138}$$

Thus, ripple sets can be used to improve the representation of the candidate item *v* for the user. Let $v \in \mathbb{R}^d$ denote the latent vector representation of the item *v*, which can be from one-hot ID, or a feature representation that incorporates attributes, depending on the dataset. First, the 1-order ripple set interest vector of the user *u* with respect to the item *v* is generated. To do this, a relevance coefficient needs to be determined for each relation in the 1-hop ripple set:

$$p_i = \text{softmax}\left(\boldsymbol{v}^\top \boldsymbol{R}_i \boldsymbol{h}_i\right) = \frac{\exp(\boldsymbol{v}^\top \boldsymbol{R}_i \boldsymbol{h}_i)}{\sum_{(h,r,t)\in\mathcal{S}_u^1} \exp(\boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{h})}, \tag{4.139}$$

where $\boldsymbol{R}_i \in \mathbb{R}^{d\times d}$ and $\boldsymbol{h}_i \in \mathbb{R}^d$ are the embeddings of relation r_i and head entity *h*. The relevance coefficient p_i can be regarded as the similarity of the item $\boldsymbol{v}$ and $\boldsymbol{h}_i$ in the relation r_i. The user's interest representation based on the 1-hop ripple set is turned into

$$\boldsymbol{o}_u^1 = \sum_{(h_i,r_i,t_i)\in\mathcal{S}_u^1} p_i \boldsymbol{t}_i. \tag{4.140}$$

Similarly, by replacing $\boldsymbol{v}$ with $\boldsymbol{o}_u^1$ and repeating the procedure on 2-hop ripple set, $\boldsymbol{o}_u^2$ can be obtained. The final vector representation of the user is the sum of *H* user interest vectors:

$$\boldsymbol{u} = \boldsymbol{o}_u^1 + \boldsymbol{o}_u^2 + \cdots + \boldsymbol{o}_u^H. \tag{4.141}$$

Generally, the value of *H* does not need to be large. In the three datasets in the paper, the optimal value of one dataset is set as $H = 2$, and the optimal value of the other two is set as $H = 3$. User's preference for the

item is modeled as the dot product of two vectors $\hat{y}_{uv} = \sigma(\boldsymbol{u}^\top \boldsymbol{v})$. Figure 4.27 depicts the modeling process of RippleNet. In addition, the model learns the prediction effect of user–item and the modeling effect of the knowledge graph at the same time, so that the user behavior as well as the entity and relation representation in the knowledge graph can be optimized together under an end-to-end framework.

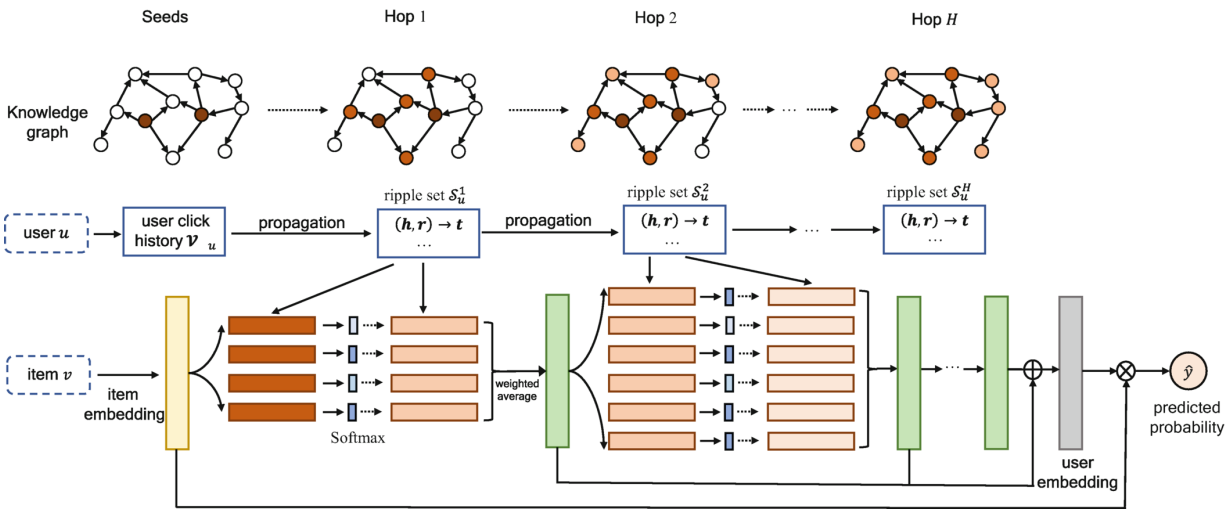


Fig. 4.27 Illustration of the RippleNet model

The KGAT Model

Considering that knowledge graph is relational data organized into graph structure, how to use graph neural network to build a recommender system based on graph knowledge is a promising task. Traditional supervised learning methods, such as factorization machine, take each sample as an independent event to predict after extracting the attribute characteristics of samples but overlook the internal relations among samples. Knowledge graph can associate samples with attributes so that samples can no longer be predicted independently. Naturally, the use of graph neural network cannot only absorb the content beyond the multiple hops on the graph structure when generating the feature representation of nodes, but also comprehensively consider the surrounding relations when predicting the user–item relations, which is called Label Smoothness. In this research direction, this section introduces a representative work—KGAT (Knowledge Graph Attention Network) [66].

In order to facilitate the establishment of an end-to-end training framework, KGAT integrates the user–item interaction bipartite graph and

the knowledge graph together to get a graph $\mathcal{G}$, which is called Collaborative Knowledge Graph (CKG). In $\mathcal{G}$, nodes contain entities, users, and items. The relations consist of the ones from the original knowledge graph plus those that reflect the user–item interactions. The KGAT framework, as shown in Fig. 4.28, includes three main layers: CKG embedding layer, attentive embedding propagation layer, and prediction layer.

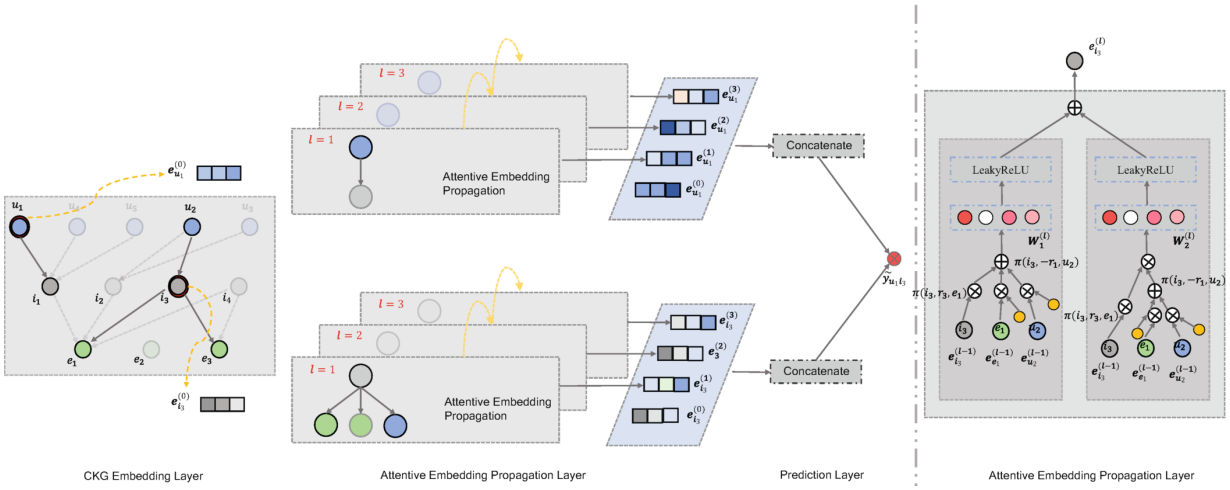


Fig. 4.28 Illustration of the KGAT model

CKG embedding layer aims to obtain the embeddings of entities and relations that preserve the graph structure. Here the TransR [37] model is employed, and it makes the triplets (h, r, t) have translation relation on the projection plane of relation *r*:

$$g(h, r, t) = \| \boldsymbol{W}_r \boldsymbol{e}_h + \boldsymbol{e}_r - \boldsymbol{W}_r \boldsymbol{e}_t \|_2^2, \tag{4.142}$$

where $\boldsymbol{W}_r \in \mathbb{R}^{k \times d}$ is the transformation matrix of relation *r*. The smaller the score of $g(h, r, t)$, the greater the probability that the triplet (h, r, t) is true. The training loss function is

$$\mathcal{L}_{\mathrm{KG}} = \sum_{(h,r,t,t') \in \mathcal{T}} -\log \sigma \left(g\left(h, r, t'\right) - g\left(h, r, t\right) \right), \tag{4.143}$$

where $\mathcal{T} = \{(h, r, t, t') | (h, r, t) \in \mathcal{G}, (h, r, t') \notin \mathcal{G}\}$, and (h, r, t') is a negative example triplet constructed by replacing the tail entity in a valid triplet; σ is the sigmoid function.

The second module is the attentive embedding propagation layer, which aims to recursively absorb the high-order connectivity on the graph and save

the important information through the graph attention network, ignoring noises. Considering the operation of a single-layer propagation, given a head node *h*, let $\mathcal{N}_h = \{(h, r, t)|(h, r, t) \in \mathcal{G}\}$ denote the set of all triplets starting with it. Then, the first-order connectivity vector representation of *h* on the graph is

$$\boldsymbol{e}_{\mathcal{N}_h} = \sum_{(h,r,t)\in\mathcal{N}_h} \pi(h, r, t)\boldsymbol{e}_t, \tag{4.144}$$

where $\pi(h, r, t)$ reflects how important the triplet is to the first-order connectivity representation of *h* and also controls how much information is propagated from the tail node *t*. It is calculated as follows:

$$\hat{\pi}(h, r, t) = (\boldsymbol{W}_r\boldsymbol{e}_t)^\top \tanh(\boldsymbol{W}_r\boldsymbol{e}_h + \boldsymbol{e}_r) \tag{4.145}$$

$$\pi(h, r, t) = \frac{\exp(\hat{\pi}(h, r, t))}{\sum_{(h,r',t')\in\mathcal{N}_h} \exp(\hat{\pi}(h, r', t'))}. \tag{4.146}$$

Finally, it is needed to aggregate the entity $\boldsymbol{h}$'s representation $\boldsymbol{e}_h$ and its ego-network representation $\boldsymbol{e}_{\mathcal{N}_h}$ to obtain the new representation $\boldsymbol{e}_h^{(1)}$ of the entity $\boldsymbol{h}$. Three types of aggregators are chosen:

1. The first is GCN Aggregator, which sums two vectors up and then passes through a non-linear transformation layer:

$$f_{\text{GCN}} = \text{LeakyReLU}\left(\boldsymbol{W}\left(\boldsymbol{e}_h + \boldsymbol{e}_{\mathcal{N}_h}\right)\right). \tag{4.147}$$

2. The second is GraphSage Aggregator, which concatenates two vectors and then passes through a non-linear transformation layer:

$$f_{\text{GraphSage}} = \text{LeakyReLU}\left(\boldsymbol{W}\left(\boldsymbol{e}_h \parallel \boldsymbol{e}_{\mathcal{N}_h}\right)\right). \tag{4.148}$$

3. The third is Bi-Interaction Aggregator, which considers two kinds of interactions, vector addition and vector element-wise product $\odot$, and then passes through a non-linear transformation layer:

$$f_{\text{Bi-Interaction}} = \text{LeakyReLU}\left(\boldsymbol{W}_1\left(\boldsymbol{e}_h + \boldsymbol{e}_{\mathcal{N}_h}\right)\right) + \text{LeakyReLU}\left(\boldsymbol{W}_2\left(\boldsymbol{e}_h \odot \boldsymbol{e}_{\mathcal{N}_h}\right)\right). \tag{4.149}$$

The above is one operation of attentive embedding propagation. To explore higher-order information, more propagation layers can be stacked:

$$\boldsymbol{e}_h^{(l)} = f\left(\boldsymbol{e}_h^{(l-1)}, \boldsymbol{e}_{\mathcal{N}_h}^{(l-1)}\right). \tag{4.150}$$

In the prediction layer, it is needed to concatenate the vectors of users and items obtained at each layer to get the final representation:

$$\boldsymbol{e}_u^* = \boldsymbol{e}_u^{(0)} \parallel \cdots \parallel \boldsymbol{e}_u^{(L)},\ \boldsymbol{e}_i^* = \boldsymbol{e}_i^{(0)} \parallel \cdots \parallel \boldsymbol{e}_i^{(L)}. \tag{4.151}$$

The user's preference for items is predicted as the inner product of the two vectors:

$$\hat{y}_{ui} = \boldsymbol{e}_u^{*\top} \boldsymbol{e}_i^*. \tag{4.152}$$

Similarly, the loss function of the recommendation prediction is also a pairwise optimization error:

$$\mathcal{L}_{\mathrm{CF}} = \sum_{(u,i,j)\in\mathcal{O}} -\log\sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right), \tag{4.153}$$

where $\mathcal{O} = \{(u,i,j)|(u,i) \in \mathcal{R}^+, (u,j) \in \mathcal{R}^-\}$ denotes the training set, and $\mathcal{R}^+$ represents the positive samples, while $\mathcal{R}^-$ is the negative samples. The joint training loss function of KGAT is

$$\mathcal{L}_{\mathrm{KGAT}} = \mathcal{L}_{\mathrm{KG}} + \mathcal{L}_{\mathrm{CF}} + \lambda \parallel \Theta \parallel_2^2, \tag{4.154}$$

where Θ is the model parameter set.

4.5.2 Joint Learning of Graph Modeling and Item Recommendation

The KTUP Model

In addition to propagating users' interests layer by layer on the knowledge graph to obtain explicit high-order connectivity information, another simpler modeling method is to expect that the learned latent vectors can directly contain the relation information in the knowledge graph, so as to strengthen the relationship between users and items. At the same time, the data contained in the knowledge graph are often incomplete, and many triplet relations are missing. The two tasks of predicting user–item and completing the knowledge graph can complement each other for mutual enhancements. Therefore, Yixin Cao et al. [6] proposed the KTUP model to jointly train the

user–item prediction module and the knowledge graph completion module through a unified translation-based framework, TransH [68]. TransH is originally a knowledge graph embedding model. In order to model many-to-many relations, it assumes that each relation owns a unique hyperplane, and two entities need to be projected to the hyperplane of a relation to determine the relation between them:

$$f(\boldsymbol{e}_h, \boldsymbol{e}_t, \boldsymbol{r}) = \|\boldsymbol{e}_h^{\perp} + \boldsymbol{r} - \boldsymbol{e}_t^{\perp}\|, \tag{4.155}$$

where $f(\boldsymbol{e}_h, \boldsymbol{e}_t, \boldsymbol{r})$ is a distance function. The smaller the value, the closer the relation between the two entities. $\|\cdot\|$ denotes the L_1-norm. $\boldsymbol{e}_h^{\perp}$ and $\boldsymbol{e}_t^{\perp}$ are the projection vectors of the entities on the plane of the relation *r*.

$$\boldsymbol{e}_h^{\perp} = \boldsymbol{e}_h - \boldsymbol{w}_r^{\top}\boldsymbol{e}_h\boldsymbol{w}_r, \tag{4.156}$$

$$\boldsymbol{e}_t^{\perp} = \boldsymbol{e}_t - \boldsymbol{w}_r^{\top}\boldsymbol{e}_t\boldsymbol{w}_r. \tag{4.157}$$

The optimization objective of TransH is to make $f(\cdot)$ of the correct triplet relation in the knowledge graph smaller than $f(\cdot)$ of the wrong one.

$$\mathcal{L}_k = \sum_{(h,r,t)\in g} \sum_{(e',r',t')\in g^-} \max\left(0, f\left(\boldsymbol{e}_h, \boldsymbol{e}_t, \boldsymbol{r}\right) + \gamma - f\left(\boldsymbol{e}_h', \boldsymbol{e}_t', \boldsymbol{r}'\right)\right). \tag{4.158}$$

As shown in Fig. 4.29, KTUP similarly models the user–item relation using TransH's framework. Suppose the user interacts with the item because of some preference *p*, and $p \in P$ is a series of predefined user preferences, and for the observed user–item interaction, there is $u + p \approx i$. Therefore, compared with the recommendation model that only needs to model the binary user–item relation, KTUP requires an additional preference induction module to predict which preference will make the user interact with the item. In general, the preference induction module has two alternative strategies: single mode (hard) and composite mode (soft). In the single mode, it is assumed that the user will only make decisions based on a single preference. In this case, the Straight-Through Gumbel SoftMax [27] is adopted for the discrete sampling to have continuous gradients during the end-to-end training. Given a user–item pair (u, i) and the preference *p*, the score of *p* is $\phi(u, i, p) = \mathrm{dot\_product}(u + i, p)$. The probability of selecting the preference *p* is the value normalized by $\log \mathrm{Softmax}$:

$$\phi(p) = \frac{\exp(\log(\pi_p))}{\sum_{j=1}^{P} \exp(\log(\pi_j))}. \tag{4.159}$$

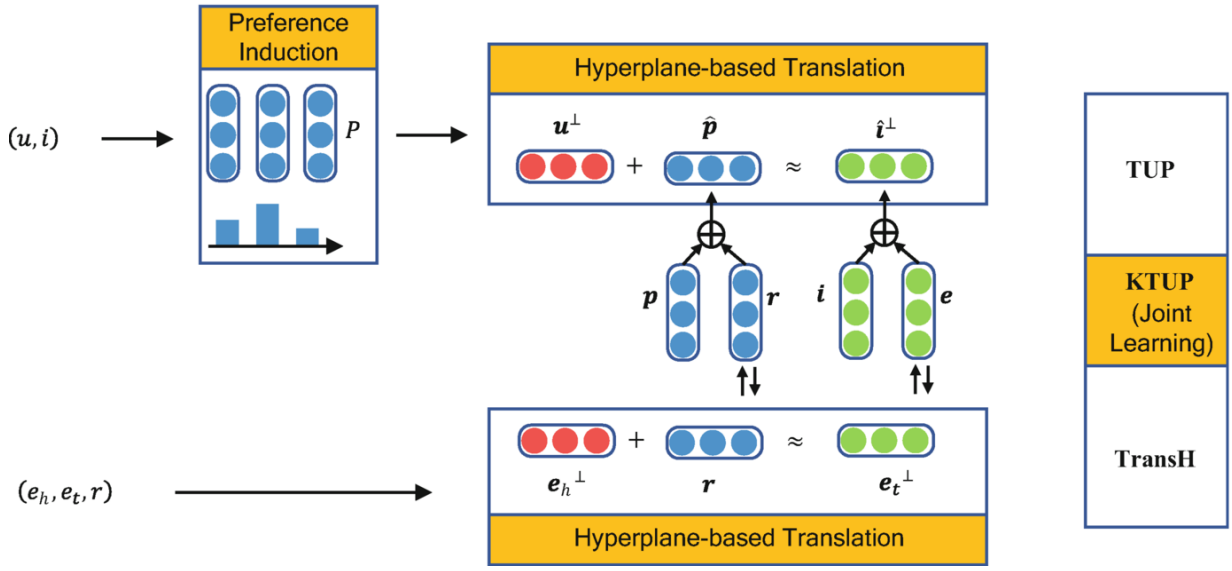


Fig. 4.29 Illustration of the KTUP model

In the composite mode, a user interacts with an item according to various preferences:

$$\boldsymbol{p} = \sum_{\boldsymbol{p}' \in P} \alpha_{\boldsymbol{p}'} \boldsymbol{p}', \tag{4.160}$$

where $\alpha_{\boldsymbol{p}'}$ is the importance of $\boldsymbol{p}'$, and it is positively correlated with $\phi(u, i, p)$. Mimicking TransH, the user–item relation can be modeled as

$$g(u, i; p) = \boldsymbol{u}^{\perp} + \boldsymbol{p} - \boldsymbol{i}^{\perp}. \tag{4.161}$$

Similarly, $\boldsymbol{u}^{\perp}$ and $\boldsymbol{i}^{\perp}$ are the vectors projected on the plane of the preference *p*:

$$\boldsymbol{u}^{\perp} = \boldsymbol{w}_p^{\top} \boldsymbol{u} \boldsymbol{w}_p, \ \ \boldsymbol{i}^{\perp} = \boldsymbol{i} - \boldsymbol{w}_p^{\top} \boldsymbol{i} \boldsymbol{w}_p. \tag{4.162}$$

For the single mode, $\boldsymbol{w}_p$ is the projection vector corresponding to the preference *p*; for the composite mode, the projection vector is also obtained by combination:

$$\boldsymbol{w}_p = \sum_{p' \in P} \alpha_{p'} \boldsymbol{w}_{p'}. \tag{4.163}$$

KTUP adopts the way of joint training to optimize the user–item recommendation module and knowledge graph completion module at the same time. The whole framework is shown in Fig. 4.29. To establish a good relationship between the two modules, it is needed to align the embeddings of the item *i* and the entity *t*, as well as the preference *p* and the relation *r*. Specifically, the recommendation module is then defined as

$$g(u, i; p) = \boldsymbol{u}^{\perp} + \hat{\boldsymbol{p}} - \hat{\boldsymbol{i}}^{\perp} \tag{4.164}$$

$$\hat{\boldsymbol{i}}^{\perp} = \hat{\boldsymbol{i}} - \hat{\boldsymbol{w}}_p^{\top} \hat{\boldsymbol{i}} \hat{\boldsymbol{w}}_p \tag{4.165}$$

$$\hat{\boldsymbol{i}} = \boldsymbol{i} + \boldsymbol{e}, (i, e) \in \mathcal{A}, \tag{4.166}$$

where $\mathcal{A} = \{(i, e)\}$ is the set of item–entity alignments. And the one-to-one mapping $\mathcal{R} \rightarrow \mathcal{P}$ from relations in the knowledge graph to user preferences is needed, and then the preference vectors can be obtained:

$$\hat{\boldsymbol{p}} = \boldsymbol{p} + \boldsymbol{r}, \ \hat{\boldsymbol{w}}_p = \boldsymbol{w}_p + \boldsymbol{w}_r. \tag{4.167}$$

The loss function of the recommendation module is

$$\mathcal{L}_p = \sum_{(u,i) \in \mathcal{Y}} \sum_{(u,i') \in \mathcal{Y}^-} -\log \sigma \left[g\left(u, i'; p'\right) - g\left(u, i; p\right) \right]. \tag{4.168}$$

And the final loss function takes into account both the errors of knowledge graph completion and recommendation:

$$\mathcal{L} = \lambda \mathcal{L}_p + (1 - \lambda) \mathcal{L}_k, \tag{4.169}$$

where $\mathcal{L}_p$ is the recommendation error, and $\mathcal{L}_k$ is the knowledge graph completion error.

The MKR Model

Training the two tasks together, item recommendation and knowledge graph embedding, can be mutually reinforcing. The collaborative behavior of many users among different items suggests that there is strong correlation behind these items, which can be used as the basis for assisting knowledge embedding. At the same time, the rich relations among items brought by knowledge graph can well alleviate the data sparsity problem in collaborative filtering and help improve the accuracy of recommendation.

Despite the strong correlation between the two tasks, there are still some differences. It is a problem worth pondering that how to balance the information sharing and differentiation between the two tasks in the end-to-end joint training framework, so as to get the best effect. Hongwei Wang et al. [59] proposed the MKR framework, which uses deep neural networks to automatically learn the information sharing and interaction between items in the recommender system and entities in the knowledge graph. Its core component is the cross and compress unit, which explicitly models high-order interactions between items and entities and automatically controls the degree of information sharing and interaction between the two tasks. As shown in Fig. 4.30, MKR consists of three main modules: recommendation module, knowledge graph embedding module, and cross and compress unit. The first two modules are bridged together by the cross and compress unit. Let the embeddings of the item and its corresponding entity in the *l*-th layer be $\boldsymbol{v}_l \in \mathbb{R}^d$ and $\boldsymbol{e}_l \in \mathbb{R}^d$, respectively, and then $\boldsymbol{C}_l$ is a cross-product matrix of $d \times d$:

$$\boldsymbol{C}_l = \boldsymbol{v}_l \boldsymbol{e}_l^\top = \begin{bmatrix} \boldsymbol{v}_l^{(1)} \boldsymbol{e}_l^{(1)} & \cdots & \boldsymbol{v}_l^{(1)} \boldsymbol{e}_l^{(2)} \\ \vdots & \ddots & \vdots \\ \boldsymbol{v}_l^{(d)} \boldsymbol{e}_l^{(1)} & \cdots & \boldsymbol{v}_l^{(d)} \boldsymbol{e}_l^{(d)} \end{bmatrix}, \tag{4.170}$$

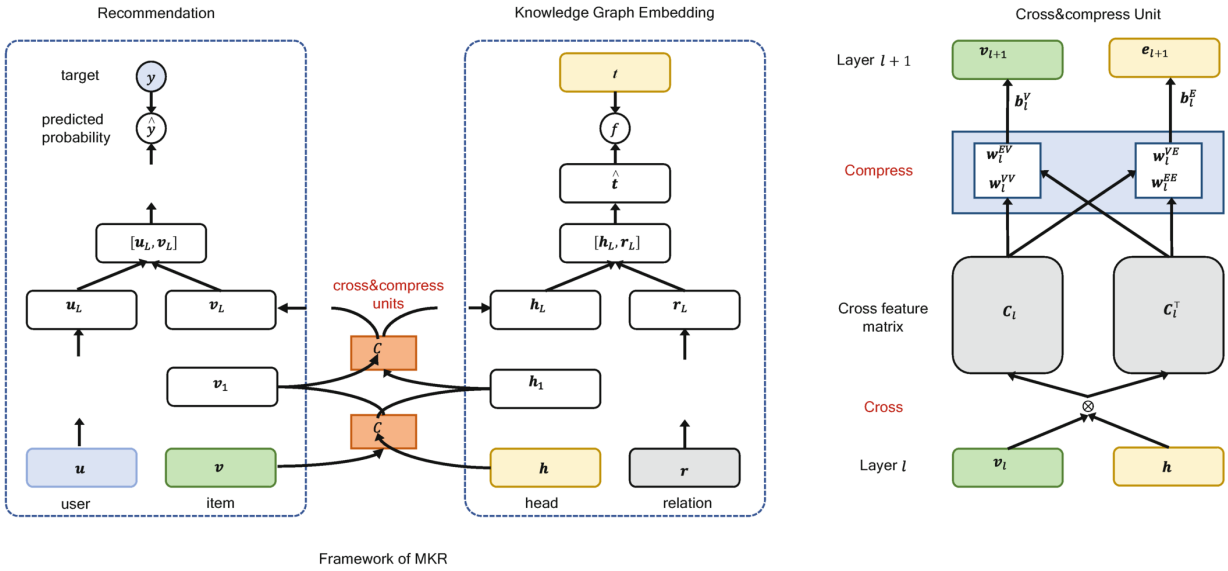
where $\boldsymbol{C}_l$ is the interaction result of the item vector and the entity vector in the *l*-th layer. The next step is to compress the interaction matrix into two vectors, respectively, representing the outputs of the item vector and the entity vector for the *l*-th layer, and also the inputs for the $l+1$-th layer:

$$\boldsymbol{v}_{l+1} = \boldsymbol{C}_l \boldsymbol{w}_l^{VV} + \boldsymbol{C}_l^\top \boldsymbol{w}_l^{EV} + \boldsymbol{b}_l^V = \boldsymbol{v}_l \boldsymbol{e}_l^\top \boldsymbol{w}_l^{VV} + \boldsymbol{e}_l \boldsymbol{v}_l^\top \boldsymbol{w}_l^{EV} + \boldsymbol{b}_l^V \tag{4.171}$$

$$\boldsymbol{e}_{l+1} = \boldsymbol{C}_l \boldsymbol{w}_l^{VE} + \boldsymbol{C}_l^\top \boldsymbol{w}_l^{EE} + \boldsymbol{b}_l^E = \boldsymbol{v}_l \boldsymbol{e}_l^\top \boldsymbol{w}_l^{VE} + \boldsymbol{e}_l \boldsymbol{v}_l^\top \boldsymbol{w}_l^{EE} + \boldsymbol{b}_l^E, \tag{4.172}$$

where $\boldsymbol{w}_l^* \in \mathbb{R}^d$ and $\boldsymbol{b}_l^* \in \mathbb{R}^d$ are parameters of the trainable compress units, which aim to compress the matrix $\boldsymbol{C}_l$ of $\mathbb{R}^{d \times d}$ into a vector of $\mathbb{R}^d$. The parameters of compress unit in each layer are different, in order to capture different degrees of information sharing between tasks through the interaction and transformation of *L* layers. For example, low-level networks need to learn more general knowledge, whose degree of sharing among different tasks is larger, while in high-level networks, task-specific knowledge needs to be gradually extracted for different tasks, so the degree

of knowledge sharing among different tasks in high-level networks is relatively smaller. To simplify the expression, let $C(\boldsymbol{v}_l, \boldsymbol{e}_l)$ be one cross and compress operation.



Framework of MKR

Fig. 4.30 Illustration of MKR

For the recommendation module in Fig. 4.30, the input is the user vector $\boldsymbol{u}$ and the item vector $\boldsymbol{v}$, which can be either a one-hot ID, or attributes, depending on the situation of different datasets. The user vector $\boldsymbol{u}$ will go through an L-layer MLP to extract the condensed vector representation:

$$\boldsymbol{u}_L = \mathrm{MLP}^L(\boldsymbol{u}). \tag{4.173}$$

The item vector $\boldsymbol{v}$ and its associated entity set $\mathcal{S}(v)$ go through L cross and compress units to extract latent vectors:

$$\boldsymbol{v}_L = \mathbb{E}_{e \sim \mathcal{S}(v)} \left[C^L(\boldsymbol{v}, \boldsymbol{e})[\boldsymbol{v}] \right]. \tag{4.174}$$

Then, the extracted user vector $\boldsymbol{u}_L$ and the item vector $\boldsymbol{v}_L$ are inner producted or concatenated through an H-layer MLP to get the prediction:

$$\hat{y}_{uv} = \sigma\left(f_{\mathrm{RS}}(\boldsymbol{u}_L, \boldsymbol{v}_L)\right). \tag{4.175}$$

Similarly, for the knowledge graph embedding module, the head entity h will go through L cross and compress units to extract the latent vector, and the relation r will go through a L-layer MLP to get the latent vector:

$$\boldsymbol{h}_L = \mathbb{E}_{v \sim \mathcal{S}(h)} \left[C^L(\boldsymbol{v}, \boldsymbol{h})[\boldsymbol{h}] \right] \tag{4.176}$$

$$\boldsymbol{r}_L = \mathrm{MLP}^L(\boldsymbol{r}). \tag{4.177}$$

Subsequently, $\boldsymbol{h}_L$ and $\boldsymbol{r}_L$ are concatenated together, followed by a *k*-layer MLP, to get the final tail entity prediction:

$$\hat{\boldsymbol{t}} = \mathrm{MLP}^K\left([\boldsymbol{h}_L, \boldsymbol{r}_L]\right). \tag{4.178}$$

The prediction that makes the knowledge graph triplet (h, r, t) true is the similarity of the predicted vector of the tail entity with its real vector, which can be obtained by inner product or cosine similarity:

$$\mathrm{score}(h, r, t) = f_{\mathrm{KG}}\left(\hat{\boldsymbol{t}}, \boldsymbol{t}\right). \tag{4.179}$$

The final training loss function is the accumulation of the recommendation function, the knowledge graph embedding function, and the regularization term.

4.5.3 Knowledge Graph Enhanced Item Representation

The DKN Model

In the news recommendation scenario, items are news articles, which have two characteristics: One is that news articles tend to have a short life cycle. For example, in the Bing News data, about 90% of the news articles are not clicked by users after two days [58]. Therefore, the traditional collaborative filtering algorithm based on ID cannot be effectively used in news recommendation. It is very crucial to understand the information of articles from the text content. Second, news articles often contain multiple knowledge entities, which condense the content of the article and enrich the information of the article from another angle. However, traditional natural language understanding models, such as Kim CNN [31], cannot capture the information of knowledge entities well. So Hongwei Wang et al. [58] proposed DKN, which integrates entity information into natural language representation model with the help of knowledge graph to generate better document representation, so as to improve the recommendation accuracy. Its core module is KCNN (knowledge-aware CNN), whose structure is shown in Fig. 4.31a. It is an extension of Kim CNN, which aligns the embedding of the word in the news title with its corresponding entity embedding and first-order neighbor entity embedding to form the 3-dimensional $d \times n$ input data. The original entity embedding is obtained by training an independent knowledge graph embedding model, such as TransE or TransH, so the

learning process of knowledge graph embedding and recommendation model are not an end-to-end unified process. To this end, DKN uses a non-linear projection layer on top of the original entity embedding, aiming to project the entity representation into the word representation space:

$$g(\boldsymbol{e}) = \tanh(\boldsymbol{M}\boldsymbol{e} + \boldsymbol{b}). \tag{4.180}$$

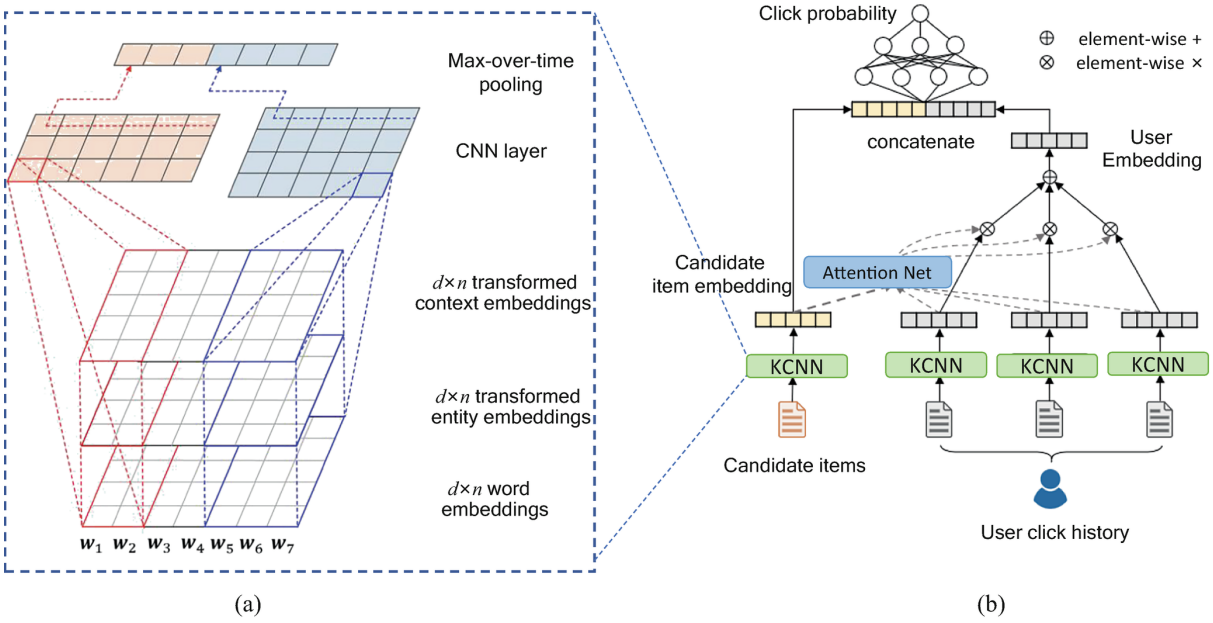


Fig. 4.31 Illustration of the overall structure of DKN and its core module KCNN. (**a**) KCNN. (**b**) DKN

Then, the Kim CNN model is applied to the 3-dimensional channel data to obtain the vector embedding $\boldsymbol{e}(t)$ of the news article *t*.

The modeling of users borrows the idea of DIN [87]. In order to distinguish the importance of different articles to different topics in the click history of the user *u*, the embedding of the news to be predicted is taken as the query vector to calculate the attention weight of each news article in the user behavior history:

$$s_{t_k^u,t_j} = \text{softmax}(\mathcal{H}(\boldsymbol{e}(t_k^u), \boldsymbol{e}(t_j))) = \frac{\exp(\mathcal{H}(\boldsymbol{e}(t_k^u), \boldsymbol{e}(t_j)))}{\sum_{i=1}^{N_u} \exp(\mathcal{H}(\boldsymbol{e}(t_i^u), \boldsymbol{e}(t_j)))}, \tag{4.181}$$

where $\mathcal{H}(\cdot)$ is the attention network, which concatenates input vectors and passes through an MLP to get the weights. The vector of the user *u* with respect to the candidate news *j* can be expressed as

$$\boldsymbol{e}(u) = \sum_{k=1}^{N_u} s_{t_k^u, t_j} \boldsymbol{e}\left(t_k^u\right). \tag{4.182}$$

In order to predict users' preferences for news, DKN combines the user vector $\boldsymbol{e}(u)$ with the candidate news vector $\boldsymbol{e}(t_j)$, followed by an MLP, to get the prediction.

The KRED Model

The core module of DKN–KCNN uses convolutional neural networks with aligned words and entities as inputs to extract hidden document vectors. This approach has two main defects. First, the computational complexity is high, and entities need to be aligned with the document and modeled with the document together. For a large number of documents without entities involved, zero vector is assigned to the corresponding position, which wastes a lot of computing power. Second, the extensibility is low. The document understanding module is based on convolutional neural network. In the current era of natural language understanding where pre-training is prevalent, KCNN is not compatible with powerful document understanding models such as BERT. In view of these considerations, Danyang Liu et al. [39] proposed KRED, a highly extensible knowledge-enhanced document representation model. For original document representation vectors of any form, such as BERT, DSSM, LDA, Kim CNN, KRED is able to inject knowledge representation in a very efficient and concise form. KRED's document representation enhancement module, as shown in Fig. 4.32, consists of three main modules: an entity representation layer, a context embedding layer, and an information distillation layer.

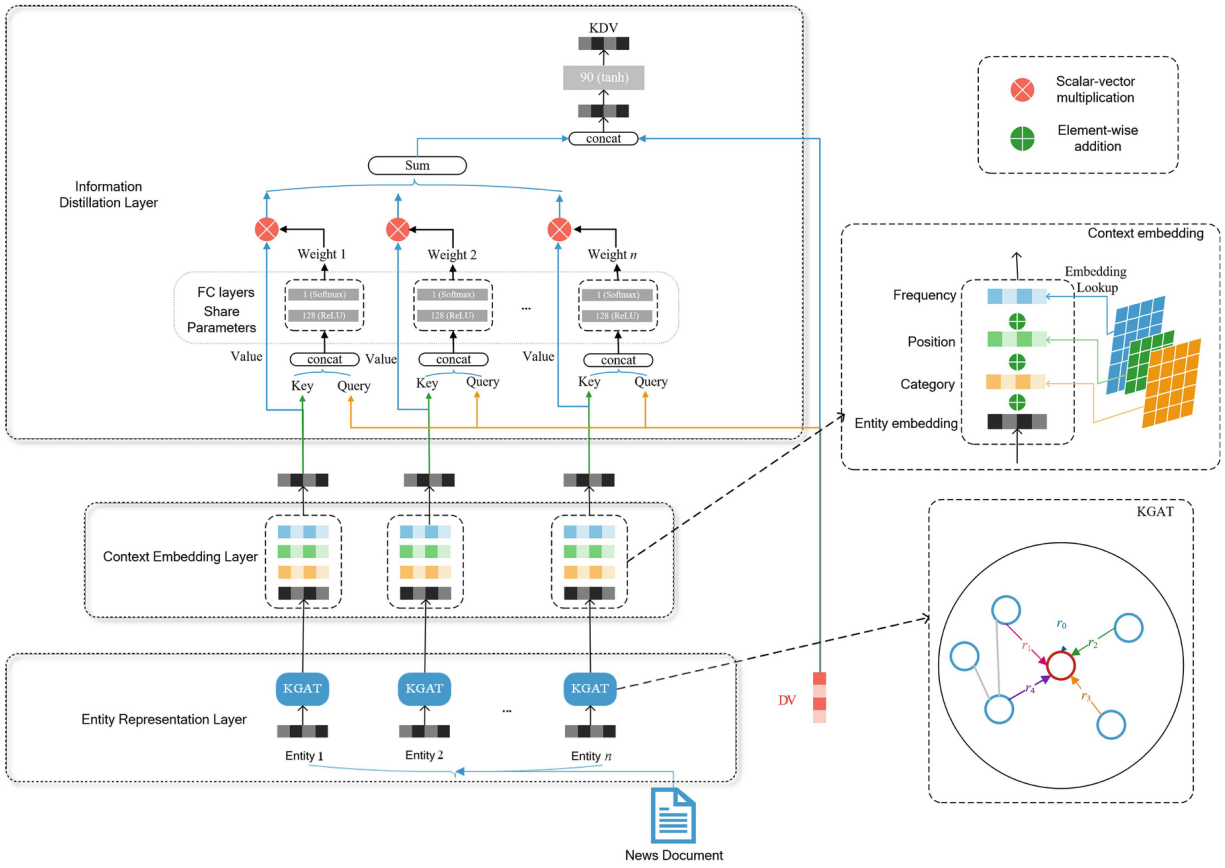


Fig. 4.32 The knowledge-enhanced document representation module in KRED

The entity representation layer exploits the idea of KGAT [66], which aggregates the first-order neighbors of the entity through the attention mechanism to enhance the representation of the entity itself:

$$\boldsymbol{e}_{\mathcal{N}_h} = \mathrm{ReLU}\left(\boldsymbol{W}_0\left(\boldsymbol{e}_h \oplus \sum_{(h,r,t)\in\mathcal{N}_h} \pi(h,r,t)\boldsymbol{e}_t\right)\right) \tag{4.183}$$

$$\hat{\pi}(h,r,t) = \boldsymbol{w}_2 \mathrm{ReLU}(\boldsymbol{W}_1(\boldsymbol{e}_h \oplus \boldsymbol{e}_t \oplus \boldsymbol{e}_t) + \boldsymbol{b}_1) + b_2 \tag{4.184}$$

$$\pi(h,r,t) = \frac{\exp(\hat{\pi}(h,r,t))}{\sum_{(h,r',t')\in\mathcal{N}_h} \exp(\hat{\pi}(h,r',t'))}. \tag{4.185}$$

The context embedding layer is designed to delineate how the entity appears in the document. For example, entities that appear in the title tend to be more important than entities that appear only in the body; entities that appear more often tend to be more important. KRED considers three types of

information: position, frequency, and category. Similar to the position and segment coding in BERT model, KRED provides embedding tables $\boldsymbol{C}^{(*)}$ for the three kinds of information and extracts corresponding embedding vectors by embedding lookup to be aggregated with the entity embedding vector:

$$\boldsymbol{e}_{\mathcal{I}_h} = \boldsymbol{e}_{\mathcal{N}_h} + \boldsymbol{C}^{(1)}_{p_h} + \boldsymbol{C}^{(2)}_{f_h} + \boldsymbol{C}^{(3)}_{t_h}, \tag{4.186}$$

where p_h, f_h, and t_h denote the position, frequency, and category of the entity *h*, respectively.

The information distillation layer is the aggregation of many entity information into a vector $\boldsymbol{e}_{\mathcal{O}_h}$. Here, the original representation vector of the document $\boldsymbol{v}_d$ is taken as the query vector, and the correlation of each entity to the document is calculated as the attention weight for weighted fusion:

$$\hat{\pi}(h, v) = \boldsymbol{w}_2 \mathrm{ReLU}(\boldsymbol{W}_1(\boldsymbol{e}_{\mathcal{I}_h} \oplus \boldsymbol{v}_d) + \boldsymbol{b}_1) + b_2 \tag{4.187}$$

$$\pi(h, v) = \frac{\exp(\hat{\pi}(h, v))}{\sum_{t \in \mathcal{E}_v} \exp(\hat{\pi}(t, v))} \tag{4.188}$$

$$\boldsymbol{e}_{\mathcal{O}_h} = \sum_{h \in \mathcal{E}_v} \pi(h, v) \boldsymbol{e}_{\mathcal{I}_h}. \tag{4.189}$$

The original representation vector of the document $\boldsymbol{v}_d$ is concatenated with the entity vector $\boldsymbol{e}_{\mathcal{O}_h}$ to obtain a knowledge-enhanced representation of the document through a non-linear layer

$$\boldsymbol{v}_k = \tanh(\boldsymbol{W}_3(\boldsymbol{e}_{\mathcal{O}_h} \oplus \boldsymbol{v}_d) + \boldsymbol{b}_3). \tag{4.190}$$

For a mature news recommender system, it is far from enough to only have a user-to-news recommendation model. Many other services are also needed, such as news-to-news recommendation, news category classification and popularity prediction, local news detection, etc. Together, these services can form a comprehensive news recommender system. The core of these services is a content-based document understanding module. Therefore, KRED uses a multi-tasking learning approach to train a unified document knowledge-enhanced model to serve different tasks, which not only eliminates the hassle of training a single model for each task, but also leverages data from different tasks to improve a single task. As shown in Fig. 4.33, KRED lists five tasks, including personalized user-to-news

recommendation, news-to-news recommendations, news category classification, news popularity prediction, and local news detection. Different tasks share KRED's document enhancement module. At the same time, each task has a small number of model parameters for its specific requirements.

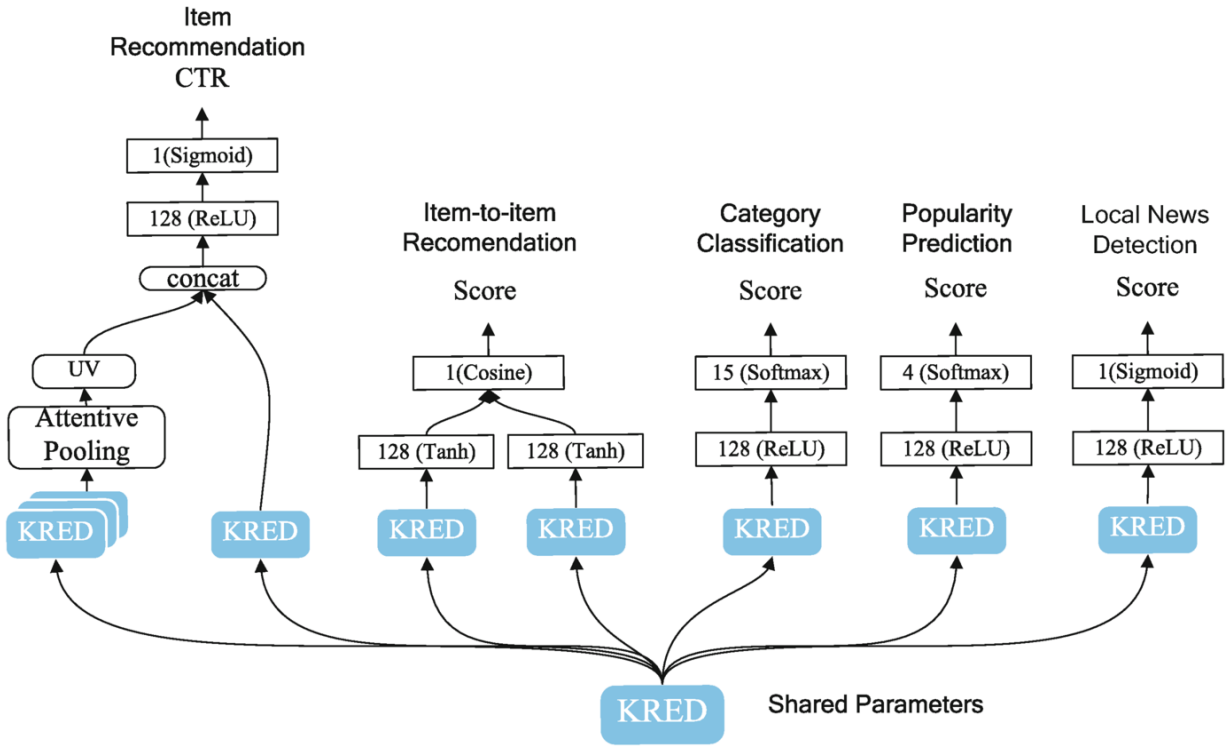


Fig. 4.33 The multi-task learning approach in KRED

4.5.4 Explainability

The KPRN Model

Knowledge graph not only brings a wealth of auxiliary information to enrich the data content, but more importantly, its structured triples rich in semantic relations can bring explainability to recommender systems. Combining the user–item interaction bipartite graph with the knowledge graph to get the collaborative knowledge graph, the path connecting users and items on the new graph structure is a recommendation candidate. For example, in Fig. 4.26, "The Green Mile – Tom Hanks – Forrest Gump" and "Back to the Future – Robert Zemeckis – Forrest Gump" can both be reasons to recommend the movie Forrest Gump to users. The most suitable path can be chosen as the reason for recommendation according to users' preferences (for example, some users prefer directors, while others prefer actors). Xiang

Wang et al. proposed the KPRN model [65] to model the paths on the collaborative knowledge graph and find out the high-quality path as the reason for recommendation. The task described by KPRN is to estimate the probability $\hat{y}_{ui} = f_\Theta(u, i|\mathcal{P}(u, i))$ of user *u* liking item *i* given user *u* and item *i* and all paths $\mathcal{P}(u, i) = \{p_1, p_2, \cdots, p_K\}$ connecting *u* and *i* on the collaborative knowledge graph. Different from other embedding-based recommendation models, $f_\Theta(\cdot)$ cannot only give scores, but also reveal recommendation reasons based on $\mathcal{P}(u, i)$.

The KPRN model mainly consists of three parts: embedding layer, LSTM layer, and weighted pooling layer, as shown in Fig. 4.34. The embedding layer is responsible for projecting three different IDs—the entity, entity type, and the relation—into the unified latent space. In the path

$$p_k = \text{Alice} \xrightarrow{\text{Interact}} \text{Shape of You} \xrightarrow{\text{SungBy}} \text{Ed Sheeran} \xrightarrow{\text{Create}} \div \xrightarrow{\text{ContainSong}} \text{I see Fire}$$

in Fig. 4.34, each entity will be represented by concatenation of three latent vectors corresponding to [entity, entity type, relation]. For example, the first entity "Alice" is represented as $\boldsymbol{x}_{\text{Alice}} = \boldsymbol{e}_{\text{Alice}} \oplus \boldsymbol{e}'_{\text{user}} \oplus \boldsymbol{r}_{\text{user\_interact}}$. The last entity "I see Fire" corresponds to a special symbol $< \text{end} >$ because it is the terminating node.

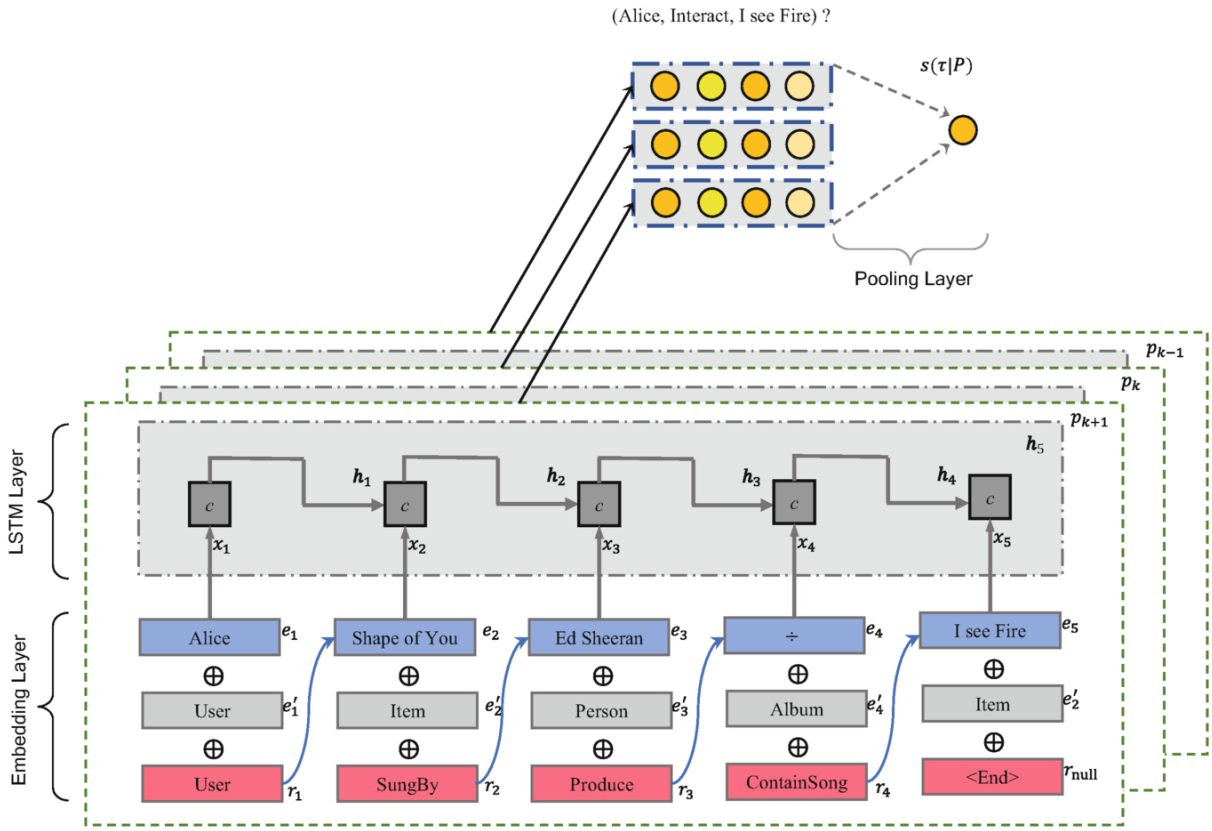


Fig. 4.34 Illustration of the KPRN model

After obtaining the entity embedding $[\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{L_k}]$ on the path p_k, the LSTM model is used to process this sequence to obtain the path embedding $\boldsymbol{p}_k = \mathrm{LSTM}([\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{L_k}])$. A two-layer MLP is then used to obtain a preference prediction score based on this path:

$$s(\tau|p_k) = \boldsymbol{w}_2^\top \mathrm{ReLU}(\boldsymbol{W}_1^\top \boldsymbol{p}_k + \boldsymbol{b}_1) + \boldsymbol{b}_2. \tag{4.191}$$

For every path p_k in $\mathcal{P}(u, i)$, a path score $s_k = s(\tau|p_k)$ is calculated. In order to distinguish the importance of different paths for predicting $\hat{y}_{ui}$, KPRN introduced a pooling layer:

$$g(s_1, s_2, \cdots, s_K) = \log\left(\sum_{k=1}^{K} \mathrm{softmax}\left(\frac{s_k}{\gamma}\right)\right), \tag{4.192}$$

where γ is the hyperparameter to control each exponential weight. The prediction result of the model is given by

$$\hat{y}_{ui} = \sigma(g(s_1, s_2, \cdots, s_K)). \tag{4.193}$$

Since the KPRN model will give a score s_k for each path p_k, by ranking the scores from high to low, the paths with high scores can be selected as the reasons for recommendation.

The PGPR Model

KPRN enumerates all the short paths that connect a bunch of users and items and then scores them. This enumeration process is expensive. The number of paths grows exponentially with the length of the path, and this scheme is usually not feasible when the knowledge map is large. Instead of enumeration and then scoring, another approach is to model the task as a path finding process on the collaborative knowledge graph. From a user, through a random walk strategy, actively selects a neighbor as the next direction of progress until a target item is reached. The path traversed during this period is an explainable path between the user and the item. Yikun Xian et al. [73] first formally described this framework and designed a reinforcement learning-based solution named PGPR (Policy-Guided Path Reasoning). The task of PGPR is described as follows: Given the collaborative knowledge graph $\mathcal{G}$, the maximum path length *K*, and the number of recommended items *N*, it can provide a recommended item set $\{i_n\}_{n\in[N]} \in \mathcal{I}$ for an input user $u \in \mathcal{U}$, so that each (u, i_n) has a path $p_k(u, i_n)$ on the collaborative graph to explain its relation.

In order to solve this task effectively, three key points should be taken into consideration. First, because this framework is a process of actively searching for recommended items without a predefined target item, the traditional reward function based on binary classification is not applicable. Therefore, it is necessary to design the reward function by combining the historical behavior and auxiliary information based on the correlation between the arrived items and users. Second, some entities have a large number of neighbors, so it is not practical to enumerate all possible paths. Therefore, it is necessary to find an effective strategy that uses reward functions as incentive to prune feasible paths. Third, the *N* items recommended to the same user should meet the diversity requirements, and similar items cannot always be recommended based on similar paths. As shown in Fig. 4.35, PGPR is a model based on reinforcement learning. Through training, an agent can meet the above three requirements. From a given user node, the agent can automatically select the appropriate path to

find good candidates. First, define the four components of this reinforcement learning method: state, action, reward, and transition.

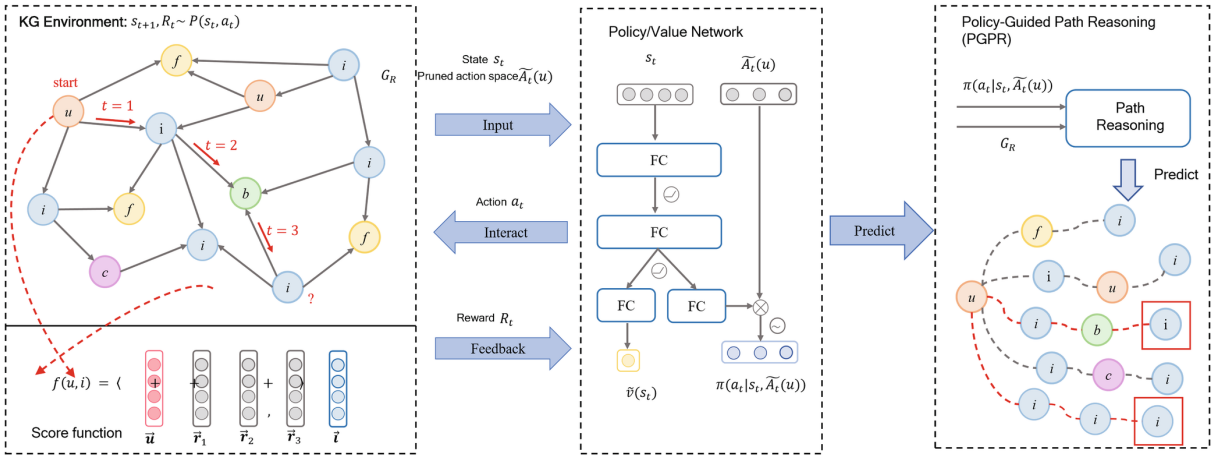


Fig. 4.35 Illustration of PGPR

State The state s_t at step *t* is defined as a triplet (u, e_t, h_t), where *u* denotes the given user node, e_t is the entity node reached at step *t*, and h_t is the history prior to step *t*. The *k*-step history is defined as all entities and relations in the past *k* steps, i.e., $\{e_{t-k}, r_{t-k+1}, \cdots, e_{t-1}, r_t\}$. The initial state is $s_0 = (u, u, \varnothing)$.

Action The action space for the state s_t is all relations starting from the node e_t (excluding those that have already appeared on the history), $A_t = \{(r, e) | (e_t, r, e) \in \mathcal{G}, e \notin \{e_0, \cdots, e_{t-1}\}\}$. Considering that out degrees of nodes follow a long-tail distribution, in order to improve the storage efficiency in the model implementation, a user-conditional scoring function $f((r, e)|u)$ is used by the authors to evaluate the potential of each edge (r, e) to the user *u*. The action space of each state can then be limited to the first K_p actions (K_p is a hyperparameter):
$\tilde{A}_t(u) = \{(r, e) \mid \text{rank}(f((r, e) \mid u)) \leqslant K_p\}$.

Reward A reward is assigned only for the terminal state $s_T = (u, e_T, h_T)$ based on another scoring function $f(u, i)$:

$$R_T = \begin{cases} \max\left(0, \frac{f(u,e_T)}{\max_{i \in \mathcal{I}} f(u,i)}\right), & \text{if } e_T \in \mathcal{I} \\ 0, & \text{otherwise} \end{cases}. \tag{4.194}$$

Transition Given a state $s_t = (u, e_t, h_t)$ and a selected action $a_t = (r_{t+1}, e_{t+1})$, the probability of transitioning to the next state s_{t+1} is

$$P[s_{t+1} = (u, e_{t+1}, h_{t+1}) | s_t = (u, e_t, h_t), a_t = (r_{t+1}, e_{t+1})] = 1. \tag{4.195}$$

Based on this Markov Decision Process (MDP) formulation, the goal of PGPR is to learn a policy π that maximizes the cumulative reward:

$$\mathcal{J}(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t R_{t+1} | s_0 = (u, u, \varnothing) \right]. \tag{4.196}$$

The authors designed a policy network and a value network to solve this reinforcement learning task through REINFORCE with baseline [53]. The input of the policy network $\pi(\cdot|\boldsymbol{s}, \tilde{\boldsymbol{A}}_u)$ is the state vector $\boldsymbol{s}$ and the binarized vector $\tilde{\boldsymbol{A}}_u$, and the output is the probability of the action. The value network $\tilde{v}(\boldsymbol{s})$ maps the state vector to a predicted reward, which will be used as the baseline in REINFORCE. The structure of the two networks is as follows:

$$\boldsymbol{x} = \text{Dropout}(\sigma(\text{Dropout}(\sigma(\boldsymbol{s}\boldsymbol{W}_1))\boldsymbol{W}_2)) \tag{4.197}$$

$$\pi(\cdot \mid \boldsymbol{s}, \tilde{\boldsymbol{A}}_u) = \text{softmax}(\tilde{\boldsymbol{A}}_u \odot (\boldsymbol{x}\boldsymbol{W}_p)) \tag{4.198}$$

$$\tilde{v}(s) = \boldsymbol{x}\boldsymbol{W}_v, \tag{4.199}$$

where $\odot$ is the Hadamard product; σ is a non-linear activation function, for which ELU is recommended; state vectors $\boldsymbol{s}$ are represented as the concatenation of the embeddings (u, e_t, h_t); the action space is $\tilde{\boldsymbol{A}}_u \in \{0, 1\}^{d_A}$; d_A represents the predefined maximum size. The policy gradient of the model is

$$\nabla_\Theta \mathcal{J}(\Theta) = \mathbb{E}_\pi[\nabla_\Theta \log \pi_\Theta(\cdot \mid \boldsymbol{s}, \tilde{\boldsymbol{A}}_u)(G - \tilde{v}(s))], \tag{4.200}$$

where *G* is the discounted cumulative reward from the state $\boldsymbol{s}$ to the terminal state s_T.

Now the scoring function for action pruning $f((r,e)|u)$ can be defined. First, $\tilde{r}_{k,j}$ is defined to indicate that on the path connecting e_0 and e_k, the first *j* relations are forward relations, and the last $k-j$ relations are backward, that is, the path $\{e_0, r_1, \cdots, r_k, e_k\}$ has the form of $\boldsymbol{e_0} \xrightarrow{r_1} \boldsymbol{e_1} \cdots \xrightarrow{r_j} \boldsymbol{e_j} \xleftarrow{r_{j+1}} \boldsymbol{e_{j+1}} \cdots \xleftarrow{r_k} \boldsymbol{e_k}$. Then the multi-hop scoring function is

$$f(e_0, e_k \mid \tilde{r}_{k,j}) = \text{dot\_product}\left(\boldsymbol{e}_0 + \sum_{i=1}^{j} \boldsymbol{r}_i, \boldsymbol{e}_k + \sum_{s=j+1}^{k} \boldsymbol{r}_s\right) + b_{e_k}. \tag{4.201}$$

For a given user–item pair (u, e), let k_e be the smallest *k* such that $\tilde{r}_{k,j}$ is valid. Then the scoring function for action pruning is $f((r,e) \mid u) = f(u, e \mid \tilde{r}_{k_e,j})$, and the scoring function for reward is derived from direct relation between the user and the item: $f(u,i) = f(u, i \mid \tilde{r}_{1,1})$.

In order to learn meaningful embeddings for entities and relations, for any pair of entities (e, e') with valid *k*-hop $\tilde{r}_{k,j}$, the probability with respect to the negative sample (e, e'') should be maximized:

$$P(e' \mid e, \tilde{r}_{k,j}) = \frac{\exp(f(e, e' \mid \tilde{r}_{k,j}))}{\sum_{e'' \text{Neg\_Sample}(\varepsilon)} \exp(f(e, e'' \mid \tilde{r}_{k,j}))}. \tag{4.202}$$

The agent guided by the policy network tends to choose the action direction with the largest cumulative reward, resulting in very similar paths. In order to improve the diversity of paths generated by agents, PGPR employs beam search to explore potential recommendation paths. At each step *t*, instead of sampling only one action according to the policy function, K_t actions with the highest probability (K_t is the hyperparameter) are put into the exploration trajectory, and only those paths whose terminal state is the item node are retained at last.

The ADAC Model

For path finding models based on reinforcement learning such as PGPR, due to the huge state and action space of collaborative knowledge graph, if the model is allowed to learn freely from a completely randomly initial state, the convergence is not fast enough, nor is it easy to converge to a good enough state. Moreover, the path found only guarantees connectivity, not necessarily qualified as a good reason for explanation. A high-quality explanation needs to include the user's preferred entities and persuasive relations in the path. Therefore, Kangzhi Zhao et al. [83] further optimized the PGPR framework. The authors believe that the key to solving these problems is how to design some mechanisms to guide and supervise the learning process of path finding. The main challenge, however, is that there is no readily labeled path reasoning data for supervised learning. Manual labeling is a very time-consuming and labor-intensive process, and it is difficult to ensure that the labeled data are complete. Therefore, Kangzhi Zhao et al. [83] proposed the ADAC (Adversarial Actor–Critic) model. First, a meta-heuristic-based demonstration extractor is used to generate a set of path demonstrations with minimal labeling efforts. These demonstrations are imperfect. Then, by optimizing both imperfect demonstration signals and the reward signal in the path finding, ADAC is able to obtain an excellent explainable path finding model with better convergence training. There are several important components to the process: a meta-heuristic-based demonstration extractor, adversarial imitation learning, and an actor–critic-based reward modeling method. The ADAC model is shown in Fig. 4.36.

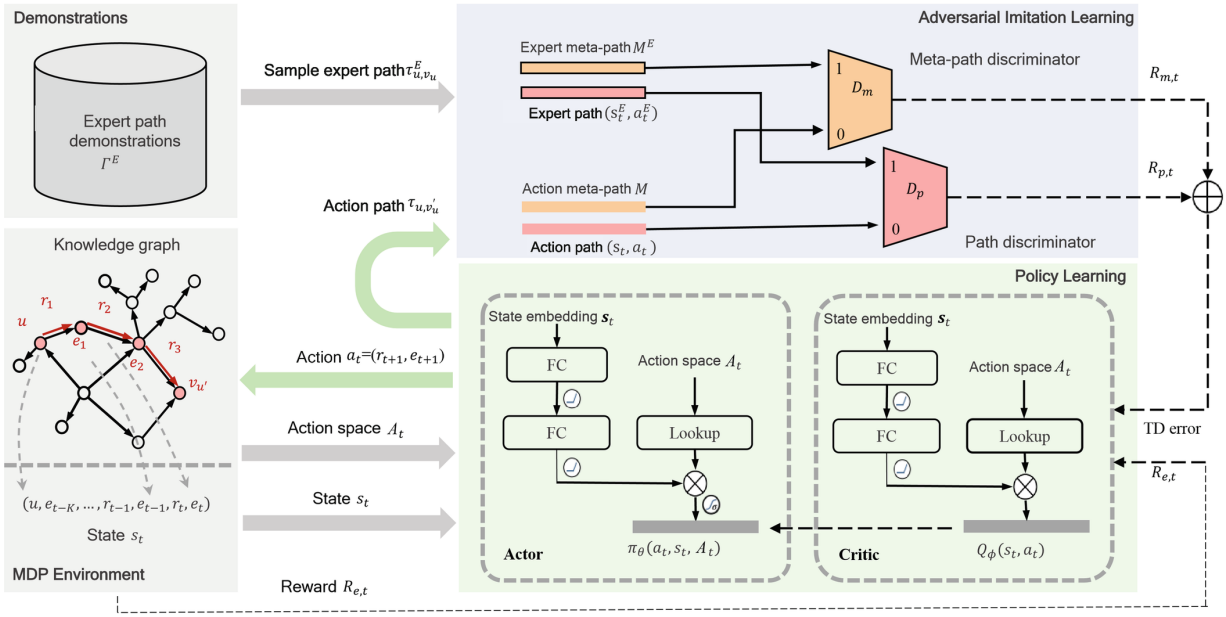


Fig. 4.36 Illustration of the ADAC model

The meta-heuristic-based demonstration extractor needs to extract a set of expert demonstration paths that connects the user and the item:

$$\Gamma^E = \left\{ \tau^E_{u,v_u} | u \in U, v_u \in V_u, \tau^E_{u,v_u} = \left[u \xrightarrow{r^E_1} e^E_1 \xrightarrow{r^E_2} \cdots \xrightarrow{r^E_k} e^E_k \right] \right\}. \quad (4.203)$$

Here, meta-heuristics means that the generated demonstration paths need to satisfy some desirable properties. These properties are threefold: easy to access, more interpretable than randomly generated paths, and able to accurately connect users with the items they are interested in. As long as these three properties are satisfied, the extracted demonstration is considered useful, even if they are imperfect and noisy. The authors proposed three heuristics, all of which can generate demonstrations satisfying these three properties: First, the shortest paths connecting users and items; second is the predefined meta-paths; third, paths that contain the entities that the users are interested in. Based on these three heuristic rules, sets of demonstrations can be generated.

Obviously, these demonstration sets are imperfect (incomplete and noisy), and the task of path finding cannot be formalized as a supervised learning process with the demonstrations as the ground-truth labels. ADAC effectively integrates the demonstration learning and the collaborative knowledge graph-based reward exploration process into a reinforcement learning framework with the help of an adversarial imitation learning module. As shown in Fig. 4.36, the main difference from PGPR is that ADAC not only has an MDP environment centered around the collaborative knowledge graph and policy learning part, but also demonstration sets to guide the learning process of the actor. The agent learns an effective policy by the actor–critic method. Paths generated by actors are fed into the adversarial imitation learning module to interact with expert paths generated by demonstration sets. The adversarial imitation learning module has two discriminators to distinguish expert paths from paths generated by the actor.

The task of an actor is to “fool” the discriminator so that it cannot tell whether a given path comes from the actor or the demonstrations. Through this adversarial learning, the actor can mimic the spirit of the demonstrations and produce high-quality paths. The goal of the critic is the reinforcement learning reward (the terminal reward, that is, whether the generated path can

successfully connect the user with the item they interact with) and two rewards obtained from fooling the discriminators.

4.6 Reinforcement Learning-Based Recommendation Algorithms

Traditional recommendation algorithms have two main characteristics: staticity and short-term focus. These characteristics provide modeling convenience but also bring some disadvantages. In this chapter, we will sequentially introduce these two characteristics and provide solutions to address the issues they present.

Traditional recommendation algorithms, including matrix factorization models, factorization machines, and other deep learning-based models introduced in this book, all attempt to model and make recommendations based on static assumptions. Specifically, these algorithms assume that the inherent distribution of user interests over different items is static. Different algorithms model the unique user interest distribution $p(v|u)$ in various ways and make recommendations based on the modeling results, i.e., $\arg\max_j p(v_j|u)$, to maximize positive user feedback, such as increasing click-through rates or purchase rates for recommended items. Researchers use predictive models to model users' preferences for different candidate items in specific recommendation scenarios. Then, personalized ranking is applied to the item list based on the current user preferences and presented to the user. Subsequently, user feedback is collected for model retraining or tuning. As shown in Fig. 4.37, the recommendation model is trained offline and deployed for recommending items to users. User feedback is collected and used for subsequent model iterations and tuning. It is worth noting that the online recommendation model has no direct perception of real-time user feedback, and the impact of the recommendation algorithm on users is limited to a single recommendation event.

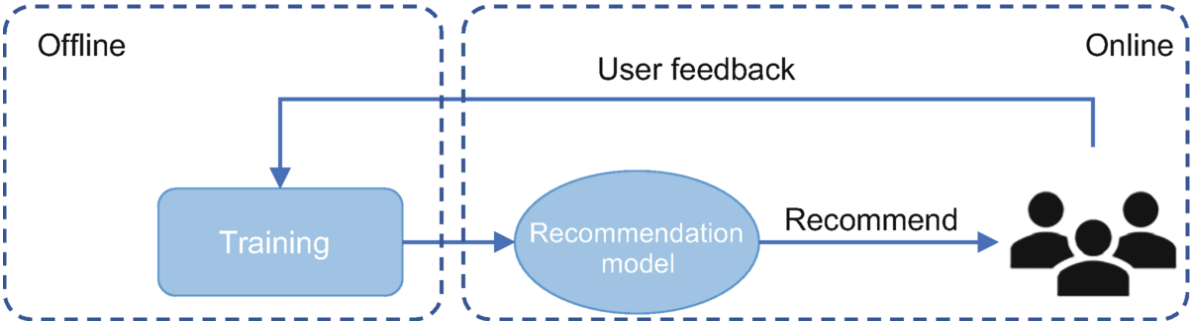


Fig. 4.37 Traditional Recommendation Algorithm Process

In this approach, traditional recommendation algorithms are characterized by their static nature, assuming that the recommendation process does not influence the user. The static assumption confines the recommendation decision-making within a framework of static modeling, reducing the complexity of understanding space and improving modeling efficiency and recommendation effectiveness. However, the static nature also gives rise to some issues. In reality, users are continuously influenced by the recommendation algorithm's results. Users may produce the corresponding feedback behaviors based on changes in the recommended content. For example, if a user is more interested in science fiction novels, the recommendation system will continuously suggest items in the science fiction category. This is reasonable for the recommendation algorithm, as historical data tell it that this user's interest is relatively fixed. However, this may cause user fatigue, resulting in a decline in interest in these recommended contents, leading to lower click-through rates or conversion rates. In this case, a static recommendation model must be retrained and fine-tuned using the latest data to adapt to the changing user interests. As recommendation systems are increasingly widely used, the impact of recommendation algorithms on users becomes more significant, which can lead to recommendation algorithms falling behind users' dynamically changing needs and thus giving rise to subsequent dynamic modeling solutions.

Due to the optimization goal of the model and algorithm for a single recommendation, this creates a short-term characteristic of the recommendation behavior. This characteristic makes it difficult for recommendation algorithms to adjust based on real-time user feedback, neglecting the long-term impact of the recommendation algorithm on users. As described in the previous example, if the recommendation algorithm keeps recommending the same type of product, it is easy to cause user

mental fatigue, leading to a long-term decrease in the effectiveness of the recommendation algorithm. As shown in Fig. 4.37, to compensate for this deficiency, traditional recommendation algorithms must frequently update the model offline, even retraining it. Another example is that the goal of many recommendation algorithms is to optimize user purchase conversion rates. User purchase behavior is often cautious and usually results from the combined effects of multiple recommendation results. In such cases, purely optimizing the accuracy of a single recommendation may not achieve the goal of conversion rate optimization.

To address the deficiencies of traditional recommendation algorithms, researchers have proposed Interactive Recommender Systems (IRS) [61, 85]. As shown in Fig. 4.38, the IRS differs from traditional recommendation systems in several aspects:

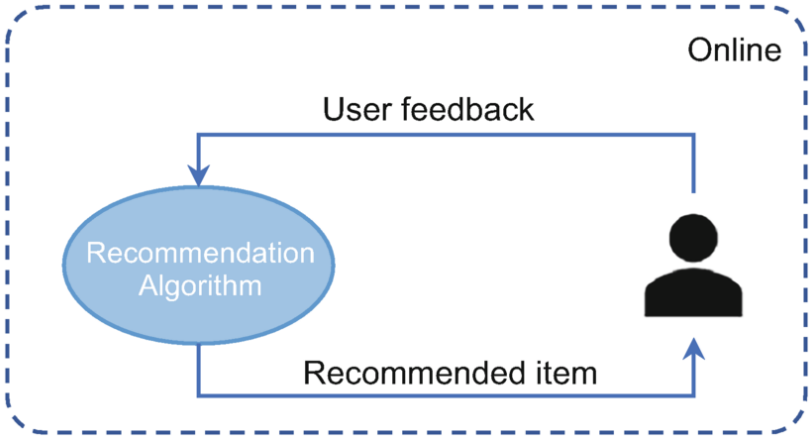


Fig. 4.38 Interactive recommender system

- IRS often operates in an online process, adjusting the next recommendation behavior based on real-time user feedback (dynamic nature). The system assumes that each recommendation action will have an impact on the current user and includes it as part of the state information to determine the next recommended items.
- Furthermore, IRS optimizes for long-term objectives (long-term nature), such as the user's purchase conversion rate and the user's active duration on the recommendation platform, over multiple recommendation interactions. These optimization objectives require the recommender system to perform multiple recommendation actions.

As a carrier of intelligent algorithms, IRS models users as the external environment and different items as choices for interactions with the environment. The real-time user feedback is modeled as a reward provided by the environment for the current recommendation action. These fundamental elements constitute the basis of IRS. There are two main categories of interactive recommendation algorithms: one based on the assumption of short-term interactive behavior, known as multi-armed bandit recommendation algorithms, and the other considering both the interaction and long-term impact, known as reinforcement learning-based recommendation algorithms. The following sections will present detailed introductions of these algorithms.

4.6.1 Multi-armed Bandit-Based Recommendation Algorithms

To address the interactive nature and balance the trade-off between exploration and exploitation in the recommendation process, some early research works used Multi-armed Bandit (MAB) algorithms to model interactive recommendation systems [7, 34, 61, 79]. As shown in Fig. 4.39, a multi-armed bandit is a common amusement facility in a casino. The basic interaction involves users choosing one arm from n arms, representing action a_i, and each arm provides a corresponding reward $r(a_i), i \in [1, 2, \cdots, K]$.

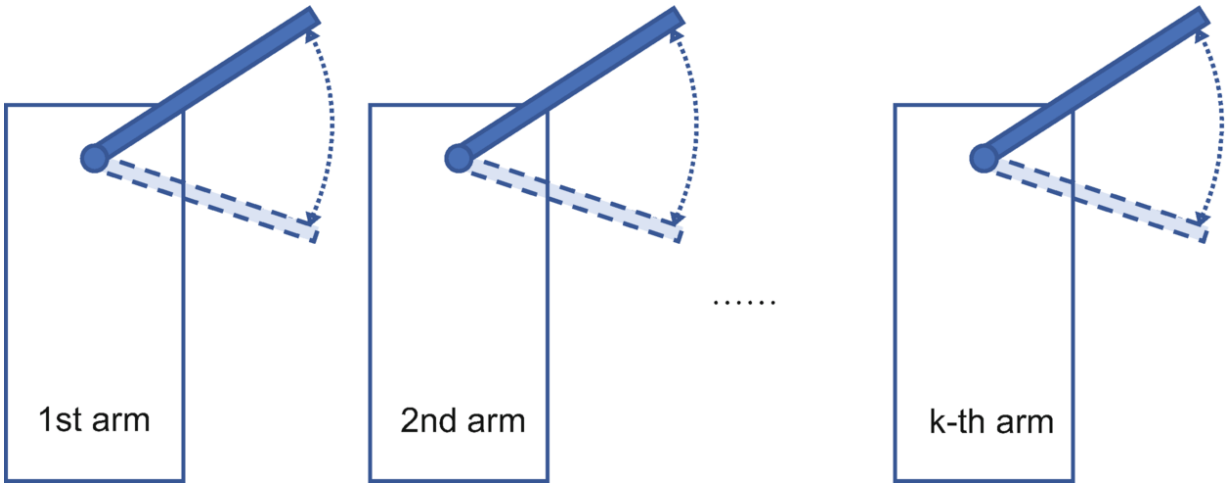


Fig. 4.39 Illustration of a multi-armed bandit

The multi-armed bandit-based recommendation algorithm treats the recommendation process as a game where users continuously choose arms of a slot machine. Different options correspond to different arms, and each

recommendation can be seen as selecting one arm from the available arms for interaction, i.e., selecting one item from the candidate set as the recommended result. User feedback corresponds to the reward obtained from choosing a particular arm. It models user preferences on a per-user basis and uses a linear function to learn and adjust the user preference continuously as the user interacts with the slot machine, optimizing the overall cumulative reward.

When using the multi-armed bandit-based recommendation algorithm to make decisions initially, there are no direct data or experience about the reward distribution $r(a_i)$ of the different arms of the slot machine. This requires addressing the exploration and exploitation problem. The exploration process involves trying actions that have not been or have been infrequently tried in the past to gather information about the reward distribution of different arms, in order to better select the arm that is most favorable in the current state. The exploitation process, on the other hand, uses existing empirical data to make the optimal choice for the current interaction.

The algorithm simultaneously performs the exploration and exploitation processes, and its optimization goal is to start the interaction from a completely unknown state and maximize the overall cumulative reward.

A simple multi-armed bandit algorithm uses previous exploration experience to directly select the arm with the highest average reward based on historical data statistics for interaction. In other words, the algorithm records the reward values of recommending different items to users and makes the next recommendation decision based on the average reward information of each item in the past. This basic multi-armed bandit algorithm adopts the ϵ-greedy exploration strategy, where with a probability of $(1-\epsilon)$, it selects the item with the highest historical reward as the next recommended decision, or with a probability of ϵ, it randomly selects another item for recommendation. This approach is straightforward to implement but utilizes limited information, resulting in a relatively low theoretical performance bound.

Next, we will introduce the recommendation algorithm based on Thompson Sampling [7]. This algorithm treats past experience observations as $D = \{(x_i, a_i, r_i)\}$, where the reward distributions are modeled using a parameterized likelihood function $P(r|a, x, \theta)$, where θ represents the function's parameters. Based on these parameters and given a prior

distribution $P(\theta)$ for θ , we can calculate the posterior distribution of these parameters as $P(\theta|D) \propto \prod P(r_i|a_i, x_i, \theta)P(\theta)$.

From a practical perspective, the actual rewards are the outputs of a stochastic function involving the arm interaction a_i , context information x_i , and the unknown true parameters θ^* . Ideally, the algorithm's primary objective is to select arms that maximize the expected return $\max_a E(r|a, x, \theta^*)$.

Of course, θ^* is unknown. If the algorithm only aims to maximize immediate rewards ("exploitation" process), it only needs to select the action that maximizes the expected return, which can be expressed as $\arg\max_a E(r|a, x) = \int E(r|a, x, \theta)P(\theta|D)\mathrm{d}\theta$.

However, in the "exploration/exploitation" scenario, according to the probability matching hypothesis (sampling actions with probabilities proportional to their probability distribution), the action obtained by randomly sampling from the probability distribution will be the optimal one. In other words, the action a will be sampled according to the probability

$$\int I\left[E(r \mid a, x, \theta) = \max_{a'} E(r \mid a', x, \theta)\right] P(\theta \mid D)\mathrm{d}\theta. \tag{4.204}$$

Here, $I(\cdot)$ is the indicator function. It is worth noting that the indicator function does not need to be computed explicitly. It can be obtained based on the posterior distribution $P(\theta|D)$ after sampling θ according to Algorithm 1. In this case, we only need to modify the operation of "pulling arms" to "recommend corresponding items" to apply it to the recommendation scenario.

Algorithm 1 Thompson Sampling-Based Recommendation Algorithm

Next, let us illustrate a specific example of the multi-armed bandit recommendation algorithm. Suppose the multi-armed bandit is a *K*-armed Bernoulli bandit, where each arm corresponds to a lever, and in the context of the recommendation system, each arm represents a different item. In other words, the recommendation algorithm can recommend K different items to the user each time. The reward of the *i*-th arm follows a Bernoulli distribution with a mean θ_i^*. When the *i*-th arm is selected, corresponding to

recommending the *i*-th item, the reward probability distribution is defined as follows:

$$f(r;\theta_i^*) = \begin{cases} \theta_i^*, & \text{if } r = 1, \\ 1-\theta_i^*, & \text{if } r = 0. \end{cases} \tag{4.205}$$

Since the Beta distribution is the conjugate distribution of the Bernoulli distribution, we can use the Beta distribution to model the mean reward of each arm as follows:

$$\theta \sim \text{Beta}(\alpha,\beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)} \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}. \tag{4.206}$$

Here, α and β represent the number of successes and failures observed in $(\alpha+\beta)$ Bernoulli trials, respectively. Therefore, Algorithm 1 can be adjusted to Algorithm 2 to address the recommendation problem in the context of the K-armed Bernoulli bandit scenario.

Algorithm 2 Thompson Sampling Algorithm for Bernoulli Bandit

Although the models based on the multi-armed bandit consider the interactive nature, they have a limitation: They assume in advance that the user preferences will remain unchanged during the recommendation process, which is not realistic. The multi-armed bandit models do not capture the dynamics of user preferences [85]. However, modern interactive recommender systems require understanding the dynamic changes in user preferences to optimize long-term rewards.

4.6.2 Introduction to Reinforcement Learning

This section introduces the fundamentals of reinforcement learning (RL). RL consists of two essential elements: the agent and the environment. The agent learns how to better achieve its optimization objectives through interactions with the environment, and these objectives are often long term. The agent's decisions have continuous effects on the environment, which requires the agent to optimize its actions continuously. For example, in a recommendation system, the optimization objectives could include the user's

final purchase conversion rate after multiple recommendations or the overall time users spend on the recommendation platform.

Reinforcement learning assumes a dynamic environment, meaning that the environment can change, especially after certain recommendation actions are taken. Users might change their interests and behaviors based on the recommended content.

Reinforcement learning has demonstrated significant achievements and potential in decision-making and long-term planning in dynamic environments. In 2014, DeepMind initiated the AlphaGo project, which used RL to achieve high-level artificial intelligence in the game of Go. AlphaGo defeated professional Go player Lee Sedol 4-1 in a five-game match and also defeated the world's top-ranked Go player Ke Jie in 2017, earning the title of professional Go nine-dan by the Chinese Weiqi Association. In 2020, Microsoft's research team released Suphx, n RL-based AI algorithm that reached a tenth-dan level on a Japanese mahjong platform. These powerful AI algorithms all demonstrate the ability of RL to optimize decision-making within large solution spaces.

Returning to the recommendation scenario, for each user recommendation, the recommendation system can choose one out of K items. If the system aims to maximize the total number of clicks over T consecutive recommendations, the feasible solution space becomes K^T, which is much larger than the number of choices in a single recommendation. Solving in such a vast space poses significant challenges and optimization difficulties. Therefore, researchers have proposed sampling- and approximation-based methods for optimization, such as Monte Carlo sampling and RL-based on deep learning. These techniques have been widely applied in RL algorithms.

In the following sections, this chapter will sequentially introduce the core concepts and key techniques of reinforcement learning in the context of recommendation systems.

4.6.3 Reinforcement Learning-Based Recommendation Algorithms

Reinforcement learning (RL) typically models the entire system using Markov Decision Processes (MDP), represented mathematically as $M = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma\}$. RL is often used for decision-making tasks, rather than the prediction and recognition tasks commonly seen in supervised

learning. The elements of an MDP include states, actions, and corresponding rewards. RL trains an optimal policy to maximize the accumulated reward of the agent over a certain period of time.

In interactive recommendation scenarios, the user and the recommendation system engage in an interaction sequence over a period of time, recording a series of recommended items and the corresponding feedback. Figure 4.40 illustrates a typical interaction process between a user and an interactive recommendation system. Throughout the process, the recommendation system corresponds to the agent in RL, while the user corresponds to the environment. Generally, the user representation and context information are considered as states $\boldsymbol{s}_t \in \mathcal{S}$. At time *t*, the recommendation system uses the state $\boldsymbol{s}_t$ provided by the environment to recommend an item to the user, where this recommendation action is referred to as $a_t \in \mathcal{A}$. The user, as part of the system environment, provides feedback to the recommendation system, including real-time rewards $r_t \in \mathcal{R}$ (e.g., clicks, conversions, time spent, etc.) and the new state $\boldsymbol{s}_{t+1}$. This interaction process continues in a loop until time *T*, which can be defined as the user leaving the recommendation process or ending the current visit session.

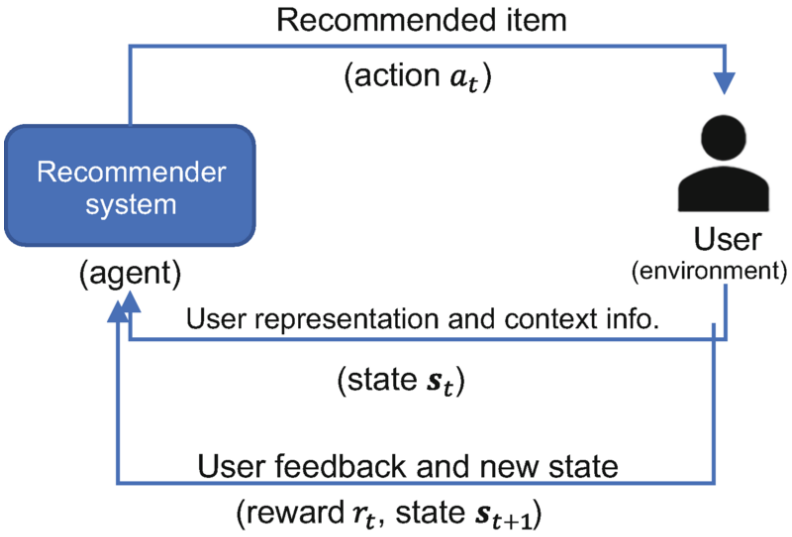


Fig. 4.40 An illustration of reinforcement learning-based recommendation

Next, we will provide a detailed explanation of several important concepts in the MDP system and their specific meanings in the context of

recommendation scenarios:

- State $\boldsymbol{s}_t \in \mathcal{S}$ represents the user's interest latent vector and contextual information at time *t*. It is usually composed of a low-dimensional real-valued vector. In general, each user can have their unique user vector, for example, the sum of *N* item representation vectors that the user has visited in the past, $\boldsymbol{s}_t = \sum_{j=1}^{N} \boldsymbol{v}_j$, or vector concatenation $\boldsymbol{s}_t = \left[\boldsymbol{v}_1^T, \cdots, \boldsymbol{v}_N^T\right]^\top$, where $\boldsymbol{v}_j$ is the feature vector of the item the user has accessed. Contextual information generally includes other information about the user's current visit, such as time representation, browser information, and other client features, which can assist the recommendation algorithm in making more reasonable recommendations.
- Action $a_t \in \mathcal{A}$ represents the recommendation algorithm choosing an item to recommend to the current user at time *t*. The action space $\mathcal{A}$ contains all the items available for recommendation. Some literature [84] defines the algorithm as recommending a whole page of items, $a = \{a^1, a^2, \cdots\}$, meaning that a single action can include more than one item. This setting is also common in real scenarios, such as recommending multiple items' content on one page, which also poses new challenges for the algorithm.
- Reward r_t is generated by the environment's reward function $\mathcal{R}(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ at time *t*, which defines the gain of the agent corresponding to the action taken in a particular state. It is also the optimization objective of the reinforcement learning-based recommendation algorithm. Generally, the reward function can be computed based on certain indicators on the environment side. For example, if the system records user purchase behavior as 1 and other non-purchase behavior as 0, then the accumulated reward of the recommendation algorithm over the *T* interactions is defined as

$$R = \sum_{j=0}^{T-t} \gamma^j r_{t+j}, \tag{4.207}$$

  where γ is the discount factor representing the importance attached to long-term rewards in the computation of cumulative rewards. The design of the reward function is a crucial part of the reinforcement learning-based recommendation algorithm, defining the performance measure that the

algorithm designer cares about. Besides the purchase conversion count mentioned in this paragraph, the reward function can also be defined as the total number of user clicks within time $0 \sim T$, the total time user spent on the platform, etc., according to the business needs.

- Transition probability $\mathcal{P} := \Pr(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, a_t)$ represents the probability of user-side state transition. In other words, it describes the probability of the environment state changing to a new state $\boldsymbol{s}_{t+1}$ given the recommendation item a_t provided by the recommendation algorithm at time *t*.
- Discount factor $\gamma \in [0, 1]$ is a real number between 0 and 1, representing the agent's consideration of long-term rewards in the computation of accumulated rewards. When $\gamma = 0$, from the formula of accumulated rewards, it can be observed that the agent only cares about the most recent reward and ignores future rewards, leading the agent to treat accumulated rewards in a near-greedy manner, optimizing short-term recommendation behavior. In this case, the reinforcement learning algorithm degenerates into a multi-armed bandit algorithm. On the other hand, when $\gamma = 1$, the algorithm takes into account the rewards in the current and future time periods. When $T \to \infty$, the agent tends to extend the interaction time infinitely rather than optimizing cumulative rewards in a finite time, which is not desirable. Therefore, the algorithm designer generally sets the discount factor to a value between 0 and 1, $0 < \gamma < 1$.

4.6.4 Modeling and Optimization of Deep Reinforcement Learning

After defining the MDP, the next step is to model the decision-making process for the recommendation algorithm based on reinforcement learning. In real-world recommendation scenarios, there are numerous and diverse items available for recommendation, resulting in a vast user state space and a particularly complex action space for recommendation behavior. To address this challenge, researchers have widely adopted deep neural networks for their powerful modeling and generalization capabilities in deep learning-based recommendation algorithms. These methods are collectively referred to as deep reinforcement learning-based recommendation algorithms.

Next, we introduce an actor–critic-based reinforcement learning recommendation algorithm. As shown in **Fig. 4.41**, this algorithm consists of two main modules: the actor network and the critic network, which are commonly used in other reinforcement learning domains as well. The actor

network is responsible for executing the recommendation operations and generating actions by directly interacting with the environment. The critic network assists the actor during training by evaluating the expected value of its actions based on the current state and actions generated by the actor. This helps the actor network to learn and improve. At the same time, the critic network also optimizes its own value evaluation ability based on the records generated from interactions between the agent and the environment. We will now introduce these two modules separately.

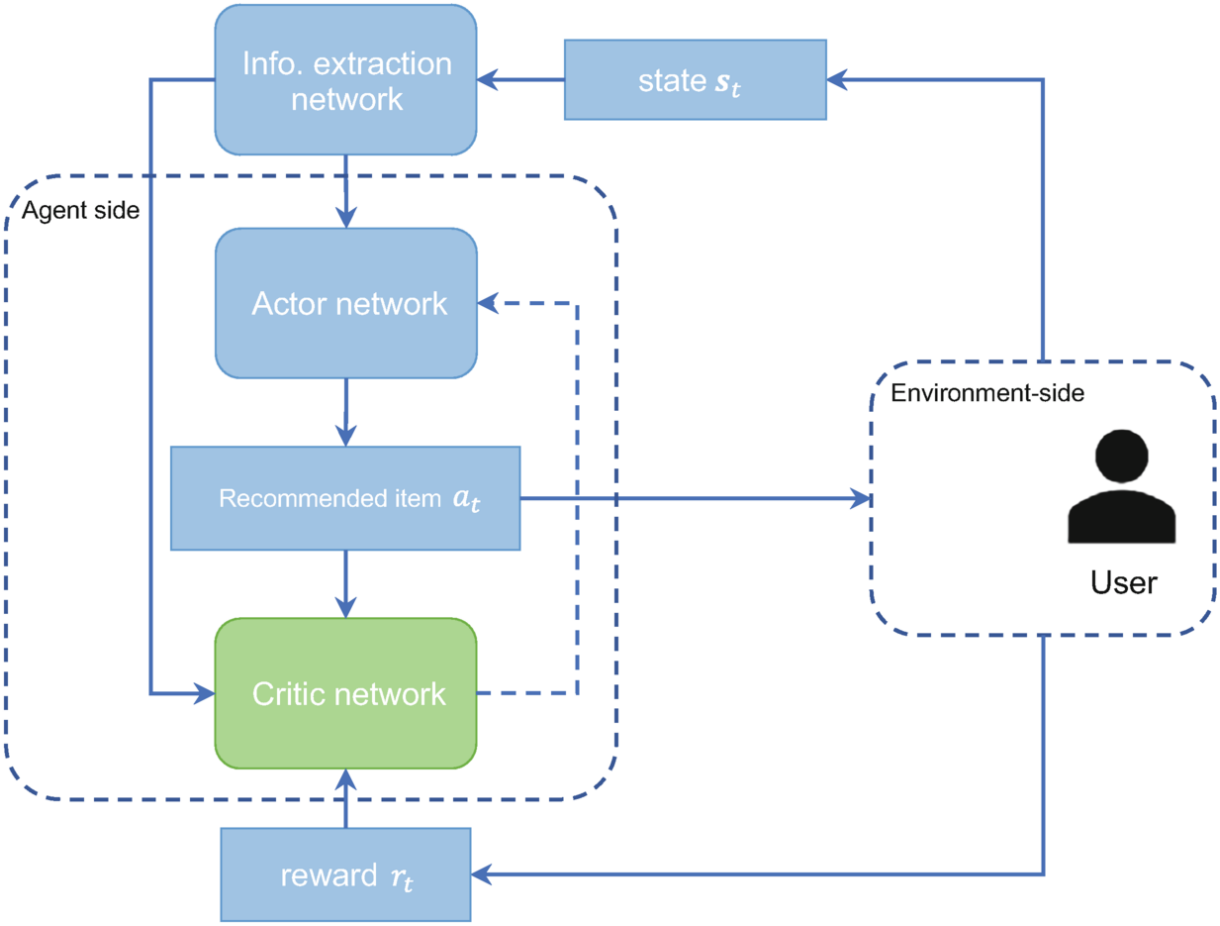


Fig. 4.41 Reinforcement learning recommendation algorithm based on actor–critic networks

In general, the actor network and the critic network share a part of the network called the information extraction network $f_\theta(\cdot)$. This network processes the raw state information into a low-dimensional representation vector in the latent space:

$$\boldsymbol{e}_t = f_\theta(\boldsymbol{s}_t). \tag{4.208}$$

As mentioned earlier, the raw state information $\boldsymbol{s}_t$ includes user interest latent vector representation and contextual information. User interest representation can be modeled and processed using the methods introduced in Sect. 4.4 for modeling user behavior sequences. Contextual information can be processed using other neural network modules, such as the wide and deep network [10]. After the information extraction network's processing, the actor network and the critic network can focus on optimizing the decision-making and evaluation parts of the network. The sharing of parameters in the lower-level network can lead to better optimization performance and has been adopted in many related reinforcement learning literature.

The actor network specifically refers to the decision-making part of the network $\pi_\phi(\cdot)$, with parameters ϕ . It outputs the action for the state $\boldsymbol{s}_t$ as follows:

$$a_t = \pi_\phi(\boldsymbol{s}_t). \tag{4.209}$$

The process of recommendation based on reinforcement learning is described in Algorithm 3. In practical use, the reinforcement learning recommendation algorithm continuously recommends *T* items or sets of items (when a single action includes multiple recommended items) for a specific user environment. The algorithm obtains immediate reward r_t and the next state $\boldsymbol{s}_t$ from the user's environment. These changing states allow the recommendation algorithm to update the recommendation plan in real time based on user-side information without the need to retrain the recommendation model using offline data. This is one of the advantages of reinforcement learning recommendation algorithms compared to traditional recommendation models. It is worth noting that the algorithm includes a trajectory set, which stores trajectories obtained from continuous *T* interactions with the user. Each trajectory consists of "transition quadruples," mainly used for training the reinforcement learning policy. Below is a detailed explanation of the process.

Algorithm 3 Reinforcement Learning-Based Recommendation Process

After completing the modeling, let us discuss the optimization methods for reinforcement learning algorithms. Recommendation algorithms based on

reinforcement learning generally adopt the online training paradigm, which means that the policy network's parameters are trained while interacting with the user environment, optimizing the cumulative reward. There are two parts to be optimized: the actor network and the critic network.

For the actor network, we introduce the policy-based optimization method. The optimization objective for the actor network is defined as

$$J(\phi) = E_{\tau \sim p_\phi(\tau)} \left[\sum_t r_t \right] \approx \frac{1}{N} \sum_i \sum_t r\left(\boldsymbol{s}_t^i, a_t^i\right), \tag{4.210}$$

where the expectation in the optimization objective is defined under the trajectory probability distribution generated by the policy defined by ϕ. Therefore, empirical averaging can be used to approximate the expected optimization objective. Next, we can derive the empirical gradient for the actor network:

$$\nabla_\phi J(\phi) \approx \frac{1}{N} \sum_i \left(\sum_t \nabla_\phi \log \pi_\phi \left(a_t^i | \boldsymbol{s}_t^i\right) \right) \left(\sum_t r\left(\boldsymbol{s}_t^i, a_t^i\right) \right). \tag{4.211}$$

Next, the gradient update for the actor network can be performed using ordinary gradient ascent:[1] $\phi = \phi + \alpha \nabla_\phi J(\phi)$.

Readers may notice that in the optimization objective of the actor network described above, for the policy at time t', it cannot influence the reward values r_t for $t < t'$. Therefore, the algorithm can optimize the expected future rewards only and take into account the effect of the discount factor. This leads to the approximation of the value function for the reward expectation.

First, define the value function as

$$V^\pi(\boldsymbol{s}_t) = E \left[\sum_{j=0}^{T-t} \gamma^j r_{t+j} \right], \tag{4.212}$$

which represents the expected cumulative reward value at time *t* under the policy π for future time steps. The critic network's parameters are denoted by ψ, and the objective of $V_\psi(\boldsymbol{s}_t)$ is to directly estimate the expected long-term cumulative reward at state $\boldsymbol{s}_t$ for the actor network:

$$\hat{V} = V_{\psi}(\boldsymbol{s}_t). \tag{4.213}$$

The optimization objective of the critic network is to estimate the expected future rewards as accurately as possible, which can be formulated as follows:

$$\begin{aligned} L(\psi) = E_{\psi}(\hat{V} - V^{\pi}) &\approx \frac{1}{N}\sum_{i}\sum_{t}\left(r_t^i + \gamma\hat{V}_{t+1}^i - \hat{V}_t^i\right)^2 \\ &= \frac{1}{N}\sum_{i}\sum_{t}\left[r_t^i + \gamma V_{\psi}\left(s_{t+1}^i\right) - V_{\psi}\left(s_t^i\right)\right]^2. \end{aligned} \tag{4.214}$$

The goal of this optimization function is to minimize the error in the critic network's estimation of state values. Therefore, we use the gradient descent formula for optimization: $\psi = \psi - \beta\nabla L_{\psi}$, where β is the learning rate. The experience gradient for the critic network can be approximated as

$$\nabla L_{\psi} \approx \sum_{i}\sum_{t}\left[r_t + \gamma V_{\psi}\left(\boldsymbol{s}_{t+1}^i\right) - V_{\psi}\left(\boldsymbol{s}_t^i\right)\right]\left(\gamma\nabla V_{\psi}\left(\boldsymbol{s}_{t+1}^i\right) - \nabla V_{\psi}\left(\boldsymbol{s}_t^i\right)\right). \tag{4.215}$$

During the training process, we can replace the cumulative expectation in the optimization objective $J(\phi)$ of the actor network with the estimated reward expectation from the critic network. Additionally, to reduce the variance in the estimation, researchers in reinforcement learning have proposed using the advantage function $A(\boldsymbol{s}t^i, at^i) = r(\boldsymbol{s}t^i, at^i) + \gamma V_{\psi}(\boldsymbol{s}t + 1^i) - V_{\psi}(\boldsymbol{s}t^i)$ instead of the original value function for the optimization objective of the policy network. Here, $r(\boldsymbol{s}t, a_t) = r_t^i$. By doing this, we can derive the new optimization gradient for the actor network as follows:

$$\nabla_{\phi}J(\phi) \approx \frac{1}{N}\sum_{i}\sum_{t}\nabla_{\phi}\log\pi_{\phi}\left(a_{t^i}|\boldsymbol{s}_{t^i}\right)\left(r_{t^i} + \gamma V_{\psi}\left(\boldsymbol{s}_{t+1}^i\right) - V_{\psi}\left(\boldsymbol{s}_t^i\right)\right). \tag{4.216}$$

Finally, we summarize the training process of the reinforcement learning recommendation algorithm in Algorithm 4. The overall algorithm iteratively optimizes the actor network and critic network for *M* epochs until they converge.

Algorithm 4 Training Process of Reinforcement Learning Recommendation Algorithm

In summary, with the increase of user data, the widening of user interest distribution differences, and the growth of item categories, recommendation systems are bound to face more challenges in terms of dynamics and long-term behavior. Both collaborative filtering and deep learning-based algorithms provide solutions to the challenges of dynamics and short-term behavior in recommendation problems. Particularly, reinforcement learning algorithms abandon the inherent assumption of static user interest distribution, incorporate real-time user feedback, and assume that the recommendation algorithm influences user behavior, providing a direction for more personalized recommendation optimization. Note that this field about deep reinforcement learning on recommender system is rapidly growing with many works publishing recent years. For readers who wish to delve deeper into this burgeoning area, they could explore comprehensive surveys like [88] that provides in-depth insights into various cutting-edge approaches.

4.7 Conclusion

In this chapter, we first introduced the relationship between collaborative filtering and deep learning and then presented various deep learning-based collaborative filtering algorithms. Leveraging cutting-edge methods from deep learning, these algorithms can significantly improve the accuracy, scalability, diversity, and interpretability of recommendation systems, offering richer technological choices for recommendation system design. However, most of these algorithms are optimized for specific problems, and there are often limitations in practical applications. Therefore, at the system design level, algorithm integration or fusion needs to be considered.

References

1. R. Agrawal and R. Srikant. “Mining sequential patterns”. In: *Proceedings of the Eleventh International Conference on Data Engineering*. 1995, pp. 3–14. https://doi.org/10.1109/ICDE.1995.380415.

2. Rianne van den Berg, Thomas N. Kipf, and Max Welling. *Graph Convolutional Matrix Completion*. 2017. https://doi.org/10.48550/ARXIV.1706.02263. URL: https://arxiv.org/abs/1706.02263.
3. Alex Beutel et al. “Latent Cross: Making Use of Context in Recurrent Recommender Systems”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM ’18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 46–54. ISBN: 9781450355810. https://doi.org/10.1145/3159652.3159727.
4. Weijie Bian et al. CAN: *Feature Co-action for Click-Through Rate Prediction*. 2020. https://doi.org/10.48550/ARXIV.2011.05625. URL: https://arxiv.org/abs/2011.05625.
5. Aleksandar Bojchevski et al. “Scaling Graph Neural Networks with Approximate PageRank”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 2464–2473. ISBN: 9781450379984. https://doi.org/10.1145/3394486.3403296.
6. Yixin Cao et al. “Unifying Knowledge Graph Learning and Recommendation: Towards a Better Understanding of User Preferences”. In: *The World Wide Web Conference*. WWW ’19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 151–161. ISBN: 9781450366748. https://doi.org/10.1145/3308558.3313705.
7. Olivier Chapelle and Lihong Li. “An Empirical Evaluation of Thompson Sampling”. In: *Advances in Neural Information Processing Systems 24. 25th Annual Conference on Neural Information Processing Systems 2011*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 2249–2257. URL: https://proceedings.neurips.cc/paper/2011/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf.
8. Tong Chen et al. “Sequence-Aware Factorization Machines for Temporal Predictive Analytics”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1405–1416. https://doi.org/10.1109/ICDE48307.2020.00125.
9. Xu Chen et al. “Sequential Recommendation with User Memory Networks”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM ’18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 108–116. ISBN: 9781450355810. https://doi.org/10.1145/3159652.3159668
10. Heng-Tze Cheng et al. “Wide & Deep Learning for Recommender Systems”. In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. DLRS 2016. Boston, MA, USA: Association for Computing Machinery, 2016, pp. 7–10. ISBN: 9781450347952. https://doi.org/10.1145/2988450.2988454
11. Wei-Lin Chiang et al. “Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 257–266. ISBN: 9781450362016. https://doi.org/10.1145/3292500.3330925.
12. Zhi-Hong Deng et al. “DeepCF: A unified framework of representation learning and matching function learning in recommender system”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33.01. 2019, pp. 61–68.

13. John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.
14. Travis Ebesu, Bin Shen, and Yi Fang. “Collaborative Memory Network for Recommendation Systems”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR ’18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, pp. 515–524. ISBN: 9781450356572. https://doi.org/10.1145/3209978.3209991.
15. Hui Fang et al. “Deep Learning for Sequential Recommendation: Algorithms, Influential Factors, and Evaluations”. In: *ACM Transactions on Information Systems* 39.1 (Nov. 2020). ISSN: 1046-8188. https://doi.org/10.1145/3426723.
16. Alex Graves, Greg Wayne, and Ivo Danihelka. *Neural Turing Machines*. 2014. https://doi.org/10.48550/ARXIV.1410.5401. URL: https://arxiv.org/abs/1410.5401.
17. Aditya Grover and Jure Leskovec. “Node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864. ISBN: 9781450342322. https://doi.org/10.1145/2939672.2939754.
18. Huifeng Guo et al. “DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI’17. Melbourne, Australia: AAAI Press, 2017, pp. 1725–1731. ISBN: 9780999241103.
19. Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *Advances in Neural Information Processing Systems 30. 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 1024–1034. URL: https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.
20. Ruining He and Julian McAuley. “Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation”. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 191–200. https://doi.org/10.1109/ICDM.2016.0030.
21. Xiangnan He et al. “LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 639–648. ISBN: 9781450380164. https://doi.org/10.1145/3397271.3401063.
22. Xiangnan He et al. “Neural Collaborative Filtering”. In: *Proceedings of the 26th International Conference on World Wide Web*. WWW ’17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 173–182. ISBN: 9781450349130. URL: https://doi.org/10.1145/3038912.3052569.
23. Balázs Hidasi et al. *Session-based Recommendations with Recurrent Neural Networks*. 2015. https://doi.org/10.48550/ARXIV.1511.06939. URL: https://arxiv.org/abs/1511.06939.
24. Geoffrey E. Hinton. “Training Products of Experts by Minimizing Contrastive Divergence”. In: *Neural Computation* 14.8 (2002), pp. 1771–1800. https://doi.org/10.1162/089976602760128018.

25. Po-Sen Huang et al. “Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data”. In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM ’13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 2333–2338. ISBN: 9781450322638. https://doi.org/10.1145/2505515.2505665.
26. Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. “FiBiNET: Combining Feature Importance and Bilinear Feature Interaction for Clickthrough Rate Prediction”. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys ’19. Copenhagen, Denmark: Association for Computing Machinery, 2019, pp. 169–177. ISBN: 9781450362436. https://doi.org/10.1145/3298689.3347043.
27. Eric Jang, Shixiang Gu, and Ben Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=rkE3y85ee.
28. Glen Jeh and Jennifer Widom. “Scaling Personalized Web Search”. In: *Proceedings of the 12th International Conference on World Wide Web*. WWW ’03. Budapest, Hungary: Association for Computing Machinery, 2003, pp. 271–279. ISBN: 1581136803. https://doi.org/10.1145/775152.775191.
29. Wang-Cheng Kang and Julian McAuley. “Self-Attentive Sequential Recommendation”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. 2018, pp. 197–206. https://doi.org/10.1109/ICDM.2018.00035.
30. George Karypis and Vipin Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392. https://doi.org/10.1137/S1064827595287997.
31. Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. https://doi.org/10.3115/v1/D14-1181. URL: https://aclanthology.org/D14-1181.
32. Yehuda Koren. “Collaborative Filtering with Temporal Dynamics”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’09. Paris, France: Association for Computing Machinery, 2009, pp. 447–456. ISBN: 9781605584959. https://doi.org/10.1145/1557019.1557072.
33. Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix Factorization Techniques for Recommender Systems”. In: Computer 42.8 (2009), pp. 30–37. https://doi.org/10.1109/MC.2009.263.
34. Lihong Li et al. “A Contextual-Bandit Approach to Personalized News Article Recommendation”. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW ’10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 661–670. ISBN: 9781605587998. https://doi.org/10.1145/1772690.1772758.
35. Jianxun Lian et al. “XDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on*

Knowledge Discovery & Data Mining. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 1754–1763. ISBN: 9781450355520. https://doi.org/10.1145/3219819.3220023.

36. Dawen Liang et al. "Variational Autoencoders for Collaborative Filtering". In: *Proceedings of the 2018 World Wide Web Conference*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 689–698. ISBN: 9781450356398. https://doi.org/10.1145/3178876.3186150.

37. Yankai Lin et al. "Learning Entity and Relation Embeddings for Knowledge Graph Completion". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 2181–2187. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571.

38. Bin Liu et al. "AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 2636–2645. ISBN: 9781450379984. https://doi.org/10.1145/3394486.3403314.

39. Danyang Liu et al. "KRED: Knowledge-Aware Document Representation for News Recommendations". In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys '20. Virtual Event, Brazil: Association for Computing Machinery, 2020, pp. 200–209. ISBN: 9781450375832. https://doi.org/10.1145/3383313.3412237.

40. Qiang Liu et al. "Context-Aware Sequential Recommendation". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1053–1058. https://doi.org/10.1109/ICDM.2016.0135.

41. Brendan McMahan. "Follow-the-Regularized-Leader and Mirror Descent: Equivalence Theorems and L1 Regularization". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 525–533. URL: https://proceedings.mlr.press/v15/mcmahan11b.html.

42. Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. https://doi.org/10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

43. Rajiv Pasricha and Julian McAuley. "Translation-Based Factorization Machines for Sequential Recommendation". In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018, pp. 63–71. ISBN: 9781450359016. https://doi.org/10.1145/3240323.3240356.

44. Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710. ISBN: 9781450329569. https://doi.org/10.1145/2623330.2623732.

45. Jiarui Qin et al. "Sequential Recommendation with Dual Side Neighbor-Based Collaborative Relation Modeling". In: *Proceedings of the 13th International Conference on Web Search and*

Data Mining. WSDM '20. Houston, TX, USA: Association for Computing Machinery, 2020, pp. 465–473. ISBN: 9781450368223. https://doi.org/10.1145/3336191.3371842.

46. Yanru Qu et al. "Product-Based Neural Networks for User Response Prediction". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1149–1154. https://doi.org/10.1109/ICDM.2016.0151.

47. Kan Ren et al. "Lifelong Sequential Modeling with Personalized Memorization for User Response Prediction". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, 2019, pp. 565–574. ISBN: 9781450361729. https://doi.org/10.1145/3331184.3331230.

48. Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. "Factorizing Personalized Markov Chains for Next-Basket Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 811–820. ISBN: 9781605587998. https://doi.org/10.1145/1772690.1772773.

49. Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. "Restricted Boltzmann Machines for Collaborative Filtering". In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, pp. 791–798. ISBN: 9781595937933. https://doi.org/10.1145/1273496.1273596.

50. Suvash Sedhain et al. "AutoRec: Autoencoders Meet Collaborative Filtering". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15 Companion. Florence, Italy: Association for Computing Machinery, 2015, pp. 111–112. ISBN: 9781450334730. https://doi.org/10.1145/2740908.2742726.

51. Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

52. Fei Sun et al. "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: Association for Computing Machinery, 2019, pp. 1441–1450. ISBN: 9781450369763. https://doi.org/10.1145/3357384.3357895.

53. R.S. Sutton and A.G. Barto. "Reinforcement Learning: An Introduction". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 1054–1054. https://doi.org/10.1109/TNN.1998.712192.

54. Jian Tang et al. "LINE: Large-Scale Information Network Embedding". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077. ISBN: 9781450334693. https://doi.org/10.1145/2736277.2741093.

55. Jiaxi Tang and Ke Wang. "Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM '18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 565–573. ISBN: 9781450355810. https://doi.org/10.1145/3159652.3159656.

56. Ashish Vaswani et al. ‘Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30. 31st Annual Conference on Neural Information Processing Systems (NIPS 2017).* Long Beach, California, USA: Curran Associates, Inc., 2017, pp. 6000–6010. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

57. Daixin Wang, Peng Cui, and Wenwu Zhu. “Structural Deep Network Embedding”. In: *tProceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1225–1234. ISBN: 9781450342322. https://doi.org/10.1145/2939672.2939753.

58. Hongwei Wang et al. “DKN: Deep Knowledge-Aware Network for News Recommendation”. In: *Proceedings of the 2018 World Wide Web Conference*. WWW ’18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1835–1844. ISBN: 9781450356398. https://doi.org/10.1145/3178876.3186175.

59. Hongwei Wang et al. “Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation”. In: *The World Wide Web Conference*. WWW ’19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 2000–2010. ISBN: 9781450366748. https://doi.org/10.1145/3308558.3313411.

60. Hongwei Wang et al. “RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems”. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM ‘18. Torino, Italy: Association for Computing Machinery, 2018, pp. 417–426. ISBN: 9781450360142. URL: https://doi.org/10.1145/3269206.3271739.

61. Huazheng Wang, Qingyun Wu, and Hongning Wang. “Factorization Bandits for Interactive Recommendation”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by Satinder Singh and Shaul Markovitch. AAAI Press, 2017, pp. 2695–2702. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14976.

62. Jizhe Wang et al. “Billion-Scale Commodity Embedding for E-Commerce Recommendation in Alibaba”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 839–848. ISBN: 9781450355520. https://doi.org/10.1145/3219819.3219869.

63. Pengfei Wang et al. “Learning Hierarchical Representation Model for Next Basket Recommendation”. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’15. Santiago, Chile: Association for Computing Machinery, 2015, pp. 403–412. ISBN: 9781450336215. https://doi.org/10.1145/2766462.2767694.

64. Ruoxi Wang et al. “Deep & Cross Network for Ad Click Predictions”. In: *Proceedings of the ADKDD’17*. ADKDD’17. Halifax, NS, Canada: Association for Computing Machinery, 2017. ISBN: 9781450351942. https://doi.org/10.1145/3124749.3124754.

65. Xiang Wang et al. “Explainable Reasoning over Knowledge Graphs for Recommendation”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii,*

USA, January 27 – February 1, 2019. AAAI Press, 2019, pp. 5329–5336. https://doi.org/10.1609/aaai.v33i01.33015329.

66. Xiang Wang et al. “KGAT: Knowledge Graph Attention Network for Recommendation”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 950–958. ISBN: 9781450362016. https://doi.org/10.1145/3292500.3330989.

67. Xiang Wang et al. “Neural Graph Collaborative Filtering”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR’19. Paris, France: Association for Computing Machinery, 2019, pp. 165–174. ISBN: 9781450361729. https://doi.org/10.1145/3331184.3331267.

68. Zhen Wang et al. “Knowledge Graph Embedding by Translating on Hyperplanes”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014, pp. 1112–1119. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

69. Jason Weston, Sumit Chopra, and Antoine Bordes. *Memory Networks*. 2014. https://doi.org/10.48550/ARXIV.1410.3916. URL: https://arxiv.org/abs/1410.3916.

70. Chao-Yuan Wu et al. “Recurrent Recommender Networks”. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM ’17. Cambridge, United Kingdom: Association for Computing Machinery, 2017, pp. 495–503. ISBN: 9781450346757. https://doi.org/10.1145/3018661.3018689.

71. Le Wu et al. “DiffNet++: A Neural Influence and Interest Diffusion Network for Social Recommendation”. In: *IEEE Transactions on Knowledge and Data Engineering* 34.10 (2022), pp. 4753–4766. https://doi.org/10.1109/TKDE.2020.3048414.

72. Yao Wu et al. “Collaborative Denoising Auto-Encoders for Top-N Recommender Systems”. In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. WSDM ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 153–162. ISBN: 9781450337168. https://doi.org/10.1145/2835776.2835837.

73. Yikun Xian et al. “Reinforcement Knowledge Graph Reasoning for Explainable Recommendation”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR’19. Paris, France: Association for Computing Machinery, 2019, pp. 285–294. ISBN: 9781450361729. https://doi.org/10.1145/3331184.3331203.

74. Jun Xiao et al. “Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 3119–3125. https://doi.org/10.24963/ijcai.2017/435.

75. Hong-Jian Xue et al. “Deep Matrix Factorization Models for Recommender Systems”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17. 2017, pp. 3203–3209. https://doi.org/10.24963/ijcai.2017/447.

76. Ghim-Eng Yap, Xiao-Li Li, and Philip S. Yu. “Effective Next-Items Recommendation via Personalized Sequential Pattern Mining”. In: *Database Systems for Advanced Applications*. Ed. by

Sang-goo Lee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 48–64. ISBN: 978-3-642-29035-0.

77. Haochao Ying et al. “Sequential Recommender System based on Hierarchical Attention Networks”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 3926–3932. https://doi.org/10.24963/ijcai.2018/546.

78. Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 974–983. ISBN: 9781450355520. URL: https://doi.org/10.1145/3219819.3219890.

79. Chunqiu Zeng et al. “Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 2025–2034. ISBN: 9781450342322. https://doi.org/10.1145/2939672.2939878.

80. Shuai Zhang et al. *Next Item Recommendation with Self-Attention*. 2018. https://doi.org/10.48550/ARXIV.1808.06414. URL: https://arxiv.org/abs/1808.06414.

81. Weinan Zhang, Tianming Du, and Jun Wang. “Deep Learning over Multi-field Categorical Data”. In: *Advances in Information Retrieval*. Ed. by Nicola Ferro et al. Cham: Springer International Publishing, 2016, pp. 45–57. ISBN: 978-3-319-30671-1.

82. Ziwei Zhang et al. “Billion-Scale Network Embedding with Iterative Random Projection”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. 2018, pp. 787–796. https://doi.org/10.1109/ICDM.2018.00094.

83. Kangzhi Zhao et al. “Leveraging Demonstrations for Reinforcement Recommendation Reasoning over Knowledge Graphs”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 239–248. ISBN: 9781450380164. https://doi.org/10.1145/3397271.3401171.

84. Xiangyu Zhao et al. “Deep Reinforcement Learning for Page-Wise Recommendations”. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys ’18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018, pp. 95–103. ISBN: 9781450359016. https://doi.org/10.1145/3240323.3240374.

85. Xiaoxue Zhao, Weinan Zhang, and Jun Wang. “Interactive Collaborative Filtering”. In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM ’13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 1411–1420. ISBN: 9781450322638. https://doi.org/10.1145/2505515.2505690.

86. Guorui Zhou et al. “Deep Interest Evolution Network for Click-Through Rate Prediction”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 – February 1, 2019*. AAAI Press, 2019, pp. 5941–5948. https://doi.org/10.1609/aaai.v33i01.33015941.

87. Guorui Zhou et al. “Deep Interest Network for Click-Through Rate Prediction”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 1059–1068. ISBN: 9781450355520. https://doi.org/10.1145/3219819.3219823.

88. Afsar M. Mehdi, Trafford Crump, and Behrouz Far. “Reinforcement Learning based Recommender Systems: A Survey”. In: *ACM Computing Surveys* 55.7 (2022), pp. 1–38. https://doi.org/10.1145/3543846.

Footnotes

1 It is worth noting that the optimization objective for the actor network is to maximize the expected cumulative reward, so we use gradient ascent.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_5

5. Recommender System Frontier Topics

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter introduces the hotspots of recommender system research, the key challenges of recommender system application, and how to achieve responsible recommendation technically. These contents may become the key of recommender system research and application in the future, so they need the continuous attention of researchers and developers.

Keywords Hotspots of recommender system research – Key challenges of recommender system application – Responsible recommendation

Recommender system is a rapidly developing area, and innovative research work is emerging every day. Only by paying attention to current or even future research hotspots, can we deeply understand the key to the research and application of recommender systems. This chapter first summarizes the research hotspots of recommendation algorithms, then analyzes the application challenges of recommender systems, and finally introduces how to implement responsible recommendation.

5.1 Recommendation Algorithm Hotspots

The previous chapters have described various recommendation algorithms and their application scenarios. However, there are still many problems to be studied in the field of recommender systems, which play an important role in the application of recommender systems. This section introduces 3 key hotspots: conversational recommendation, causal recommendation, and common-sense recommendation.

5.1.1 Conversational Recommenders

Traditional recommendation algorithms do not have much interactions with users, making it hard to capture their interests timely and effectively. Conversational recommender systems (CRSs) can get users' interests through in-depth interaction with users, which has become a new research hotspot in the field of recommender systems. The core of conversational recommenders is the online interaction between users and recommender systems, that is, through the conversational interaction process between users and recommender systems, users' feedback is obtained and then integrated into the recommendation model, in order to better understand users' interests and improve the recommendation accuracy.

Lei et al., from Nanyang Technological University in Singapore, have done a summary on conversational recommenders and proposed 4 research issues that should be paid attention to in this area [25].

Exploration–Exploitation Trade-Off in Cold-Start Scenarios

Conversational systems can easily collect users' interest information, so it can help recommender systems solve the problem of cold start. For example, Christakopoulou et al. [14] proposed a conversational restaurant recommender system that quickly learns users' interests by constantly asking them questions and then making recommendation. This method needs to consider how many questions it is appropriate to ask the user. If there are too many questions, users may lose interest in answering or even lose trust in the recommender systems. If there are too few questions, the system may have difficulty in accurately capturing users' interests, resulting in a lower recommendation accuracy. Therefore, there is a classic trade-off between exploration and exploitation in solving the cold-start problem of conversational recommenders. A common technique used to solve such

problems is the multi-armed bandit algorithm. For example, Christakopoulou et al. [14] designed an exploration–exploitation trade-off using the multi-armed bandit algorithm. First, they set the user's trait vector as the average of all users, then ask the user questions based on the multi-armed bandit algorithm, and finally update the user's trait vector based on the answers to the questions.

Question-Centered Conversational Recommendation

In this type of scenario, the conversation is initiated by the user, and the recommender system decides how to recommend items based on the user's questions and how to go forward and ask more questions. For example, in a movie recommendation scenario, the user might ask the system to recommend a recent hit comedy. At this point, the purpose of the conversation is to better understand the user's intention, such as what comedy movies the user has seen in the past, as well as preferences for something like directors, actors, years, and language, and then based on the user's answers, build the user's trait vector until the recommended movie meets the user's needs. In 2018, Li et al. [29] published a conversational movie recommendation dataset ReDial and proposed an autoencoder-based recommendation algorithm to realize conversational recommendation, which can predict users' opinions on movies based on dialogue and sentiment classification and then input user preferences into the autoencoder to make recommendation. The autoencoder method can train a neural network to reconstruct the input and estimate the unobserved user ratings in the process of reconstruction. Since the network parameters of the autoencoder do not need to be retrained for new users, the recommendation can be completed by giving a user's rating vector, so it is more suitable for the conversational recommendation scenarios where users' historical interests cannot be observed.

Strategy-Centered Conversational Recommendation

Many real-world conversations are multi-round, meaning the user may answer multiple different kinds of questions in a single conversation. For example, when recommending a restaurant to a user, it is necessary to know not only the user's preferences, but also other information like the location of the user. The core is how to determine the properties of the question to be asked, and then based on the answer to the question, decide on the next

question and what to recommend. Sun et al. [39] proposed a conversational recommendation system that includes a recommendation module, a belief tracker module, and a policy network module. The belief tracker module is used to analyze the intention of the user in the conversation, and then the policy network module is used to decide whether the next step should be to ask questions or recommend. If the decision of the policy network module is recommendation, the system will call the recommendation module and generate the recommendation result according to the user's interest reflected in the conversation and historical interest.

Dialogue Understanding and Generation

The most basic problem in a conversation is understanding exactly what the user is saying and generating natural and relevant answers. This problem is also what dialogue system researchers focus on. Different from the general dialogue system, the dialogue in the recommender system is generally based on a specific field, such as movies, music, restaurants, etc., so the knowledge related to the field is very important for the understanding and generation of the dialogue. Chen et al. [10] proposed to use knowledge graph to introduce contextual information related to items, such as film directors, actors, and genres, into the recommendation dialogue. The introduction of such information enables the recommender system to make recommendation when users do not mention specific items. Meanwhile, the introduction of such contextual information can also help improve the accuracy of the recommender system.

5.1.2 Causal Recommendation

Causal learning studies how to find and use causal relationships between variables for prediction, rather than relying only on correlation between variables. The causal relationship reveals the nature of an event, and changing the "cause" behind the event will often affect the "effect" of the event. However, correlation is often not the nature of events. Changing one event may not affect the other that is correlated. For example, association mining can find a strong correlation between yellow fingers and lung cancer, but there is no clear causal relationship between the two. Painting a person's fingers yellow does not increase their chances of getting lung cancer. In fact, the "cause" behind the yellow fingers and lung cancer is smoking, that is, there is a causal relationship between smoking and lung

cancer. Therefore, letting non-smokers smoke will increase their chances of getting lung cancer significantly. In recommender systems, the lack of causality analysis may result in the decline of the recommendation effect or the bias of the model.

First, causality will affect the model training of recommender systems. Recommender systems generally assume that the user will browse through all the items and choose the ones they like, but in practice this assumption is not true. Wang et al. [48] believe that movies are not exposed to users at random, but users select movies through a biased distribution and then make assessment within the selected movie range. For this problem, they designed a deconfounded recommender system. First, the exposure model of the movie to the user is modeled based on the user rating data, and the unobserved confounders in the system are estimated based on the model. Then the recommendation model fits the observed ratings taking into account confounders. Based on this design, the recommender system can analyze the relationship between different movie exposures and user ratings. Experimental analysis shows that, after debiasing the confounding, the recommendation algorithm shows stronger generalization ability, especially for new users, and can provide more accurate recommendation.

In addition, causality also affects the evaluation of recommender systems. When measuring the accuracy of recommender systems, offline evaluation often cannot accurately measure the recommendation effect. The reason behind is that it is impossible to intervene with users in offline evaluation, so it is difficult to calculate what users' feedback is when recommending other items. For example, in historical data, the user u purchased the item a, but in fact if the user had seen the item b, the user was more likely to purchase the item b. In this case, if the recommender system recommended the item b to the user u, it would be considered as invalid by offline evaluation. Such problem can be resolved by moving from offline to online evaluation. But online evaluation is costly and generally difficult to use widely. A simpler strategy is to adopt counterfactual reasoning [18], such as importance sampling to correct the bias in observed samples and achieve unbiased estimation.

5.1.3 Common-Sense Recommendation

As with other areas of AI, recommender systems suffer from data integrity issues, where observed data cover only a fraction of real-world situations.

Thus, even if the recommended results are reasonable within the range of observed data, the results may appear unreasonable by humans, i.e., may not conform to common sense. The recommendation that does not conform to common sense will lead to the reduction of recommendation accuracy and even affect users' trust in the recommender system. For example, many current recommender systems do not consider the conflicts between the items users purchased in the past and the recommended ones. After users have purchased TV sets, recommender systems of many e-commerce companies will continue to recommend TV sets, which is obviously not common sense. However, since there is no common-sense knowledge in the data observed by recommender systems, it is difficult for the systems to solve such problems.

Common-sense knowledge base is a key technology to solve the above problems. To solve the problem of keyword recommendation in search engines, Tsai et al. [41] proposed to combine ConceptNet and Wikipedia to associate related semantic keywords with user query keywords and then sort all related keywords. This method uses the knowledge from the common-sense knowledge base ConceptNet, so the recommendation can ensure a high degree of rationality. However, this method is difficult to extend to a wider range of recommendation scenarios, such as the problem that TV sets are repeatedly recommended in e-commerce. In addition to ConceptNet, frequently used common-sense knowledge bases include Tuple KB, Quasimodo KB, WebChild, and True Knowledge. How to use these knowledge bases to better guide the generation of recommendation lists is an important research area to be explored. At present, it is rare to find research works that combine common sense to improve the quality of recommendation. This area could be a new direction for recommender systems in the future.

5.2 Application Challenges for Recommender Systems

Recommender systems have been successful in many commercial fields, but they also face many technical challenges, including how to integrate multiple types of data to improve the recommendation accuracy, how to extend recommendation algorithms to large-scale data, how to evaluate

recommendation algorithms effectively, and how to recommend for new users or new items. This section will focus on the four key challenges aforementioned, but apparently the challenges that recommender systems face are not limited to these.

5.2.1 Multi-source Data Fusion

The multi-source data fusion problem studies how to improve the recommendation accuracy by fusing multiple types of data (also known as multiple modals). User interaction data (such as user ratings) are the most common data in the recommendation algorithm research. However, additional information such as user personal information, item attribute information, and user social relationship, which can further mine the characteristics of users and items, has also attracted more and more attention from researchers. Although there are many mature solutions to the problem of multi-source fusion in computer vision, it still needs to be further studied in the field of recommender systems. Some complicated problems in recommender systems can be solved by means of multi-source data fusion. For example, when a new user joins the system, although there are no interactive data of the user, the cold-start problem of the new user can be solved if the user's personal information can be obtained. The difficulty and key of multi-source data fusion is how to organically integrate all types of data according to the characteristics of different types of data, so as to jointly mine the characteristics of users and items, and improve the recommendation accuracy.

A common method of multi-source data fusion is to directly concatenate or sum multiple types of data (usually represented as feature vectors). Although this kind of method is simple to operate and easy to implement, it cannot achieve satisfactory results in most cases, and even in some specific cases, the recommendation accuracy is not as good as those based on rating data directly. The main reason is that simple splicing operation will make the dimension of feature vectors expand constantly, increase the difficulty for model training, and make models overfit easily. And the simple summation operation will ignore the semantic information of each type of feature before fusion, resulting in the ambiguity of feature meaning, and some key features are even covered by noise, resulting in the decrease of recommendation accuracy.

At present, the method commonly used in academia and industry is to design a neural network to realize the deep fusion of multi-source data. Attention mechanism [43] is one of the most commonly used methods to integrate features. It selects and integrates features by automatically assigning a weight to each type of feature, and the weight will change with different users or items, so that features can be integrated according to the characteristics of users or items. Ensemble learning is also often used to realize multi-source data fusion at the algorithm level. For example, a recommendation model can be trained for each type of data to obtain the corresponding recommendation results, which are then ensembled to give recommendation. Since ensemble learning integrates directly from recommendation results, there is no problem of feature meaning ambiguity. Moreover, all types of data are comprehensively considered in recommendation, which makes the model with high accuracy. Ensemble learning requires separate model design and training for each type of data, which has higher requirements on hardware resources. Especially when there are many types of data to be fused, this method consumes more hardware resources, which requires algorithm designers to choose carefully according to the actual situation.

5.2.2 Scalability

Scalability includes horizontal scalability and vertical scalability. Horizontal scalability mainly studies how to extend the recommendation algorithm to large-scale data scenarios without affecting the algorithm accuracy. Vertical scalability studies how to quickly update models and recommendation as new user or item interactions occur. The recommendation algorithm with good scalability should achieve similar accuracy and acceptable efficiency in large-scale application scenarios (such as e-commerce recommender systems with millions of users and items). Otherwise, the user experience of the recommender system will be seriously affected. Additionally, when a user generates a new interaction record, the recommendation algorithm with good scalability needs to quickly analyze the user’s interaction behavior and recommend those more in line with the user’s preferences. The methods commonly used to solve the scalability problem of recommendation algorithms mainly include clustering, dimension reduction, distributed computing, and incremental

recommendation. The advantages and disadvantages of these methods are briefly discussed below.

Clustering

Clustering algorithms improve the efficiency of recommendation by narrowing the search space of similar users or items. Recommendation algorithms that adopt clustering first divide users or items into different clusters according to their characteristics, so that the similarity of users or items within the cluster is as large as possible. When a recommendation is calculated for a user, only other users or items in the same cluster as the user or item are considered, not all users or items in the entire set. For example, if the original dataset is evenly divided into 100 clusters, then each model only needs to process 1% of the original data volume, the training or prediction of each model can be processed in parallel, and the scalability will be greatly improved. The advantage of this method is that it reduces the space of model training and prediction, but it also makes the accuracy of the recommendation algorithm easily affected by the accuracy of clustering. If the clustering algorithm cannot provide an optimal subspace, the recommendation algorithm is likely to give poor recommendation.

Dimension Reduction

Dimension reduction methods, such as singular value decomposition, improve the efficiency of recommendation by reducing the dimensions of user or item’s feature representations, which can be determined by a preset threshold. Although singular value decomposition and other methods can improve the efficiency of the algorithm by reducing the feature space of users and items, and the smaller the feature space is, the higher the efficiency of the algorithm, it also has a certain impact on the recommendation accuracy because the reduction of the feature space of users and items may lead to the loss of some useful information, which will affect the accuracy of the recommendation results. Therefore, how to balance the accuracy and efficiency of the algorithm is a key problem to be solved by dimension reduction methods such as singular value decomposition.

Distributed Computing

Commercial recommender systems need to process massive user interaction records every day, which is very difficult by single machine, no matter whether it is from the perspective of storage or computing. For example, it may take several days to complete the calculation of user and item characteristics. In order to ensure real-time performance and high efficiency of recommendation, commercial recommender systems usually adopt some distributed computing frameworks (such as Hadoop and Spark) to accelerate the extraction of user and item features, reducing the model training time from several days to just a few minutes, which greatly improves the efficiency of recommendation. Currently, distributed frameworks such as Spark have implemented common recommendation algorithms such as matrix decomposition.

Incremental Recommendation

Incremental recommendation can quickly analyze user interaction behavior and update recommendation model and results after new user interaction records are generated. When non-incremental recommendation algorithms are faced with new interaction records, they will retrain the model in combination with the original user interaction records. However, this way of training takes longer time and badly affects the real-time performance of recommendation. Incremental recommendation usually retains the features of users and items in the previous stage as historical feature information. When new interaction data are generated, the algorithm can quickly train the model based on the historical feature information and the new interaction data, so as to quickly update the recommendation results. After the model training of the current stage is completed, the historical feature information is updated with the features obtained from the training to prepare for the next incremental recommendation. Incremental recommendation is applicable to recommendation scenarios that require high real-time performance. Therefore, non-real-time recommendation scenarios usually do not require incremental recommendation.

5.2.3 Performance Evaluation

There are three main approaches to the performance evaluation of recommender systems: offline evaluation, online evaluation, and user study. The principles and challenges of these three approaches are discussed below.

Offline Evaluation

Offline evaluation is to complete the testing and evaluation of the recommender system according to some evaluation metrics in the offline environment. Offline evaluation usually uses easily accessible experimental datasets and is attractive to researchers due to its ease of implementation. However, in the process of acquiring datasets, what researchers need to pay attention to is that the datasets obtained should be able to simulate the real application scenarios faced by the recommender system. Especially for recommendation algorithms targeting at multiple application scenarios, researchers should collect as many different types of datasets as possible to cover all application scenarios, so as to ensure that the test results are comprehensive and reliable. The way of dividing dataset and the choice of evaluation metrics are also problems that need to be considered in offline evaluation. First, dataset partitioning methods are usually hold-out and cross validation. The process of hold-out is simple but easy to make the model overfit. Cross validation can evaluate the generalization ability of the model more effectively, but the process is complicated. Researchers need to choose the appropriate dataset partitioning method according to the actual scenario. Second, different evaluation indices will evaluate the performance of the recommender system from different perspectives, but usually not all evaluation metrics can achieve satisfactory results in the evaluation process. Researchers should consider the pros and cons of different evaluation metrics according to the application scenario, as well as the emphasis of the recommender system to ensure the reliability of the evaluation results. It should be noted that during offline evaluation, users' responses may vary depending on what are recommended. For example, as mentioned earlier in causal recommendation, if the list of recommendation is changed for the user, the user's choices may not be consistent with the behavior in the historical data. Therefore, offline evaluation usually cannot accurately estimate the actual effect after the recommender system is online.

Online Evaluation

Online evaluation can effectively solve the inaccuracy problem of offline evaluation. The common method of online evaluation is A/B testing: The researchers divide the tested users into two groups and recommend with two different methods. Over time, the researchers collect feedback from users in each group and compare the two methods. Because of its simple

principle and easy implementation, A/B testing has been widely used in the testing and evaluation of recommender systems. It can help researchers quickly choose the appropriate method and effectively guide the improvement direction of the recommender system. However, there are some problems with online evaluation. Because online evaluation needs to be carried out in an online environment, excessive traffic or frequent testing may affect the practicability and user experience of the system. If these problems cannot be properly solved, it will result in bias in the selection and affect the effectiveness and reliability of evaluation results. Therefore, researchers need to choose appropriate time to use online evaluation, which generally needs to be used when they are confident at the evaluation method and the evaluation environment meets certain requirements. Additionally, researchers should choose appropriate online evaluation metrics and user grouping methods according to the situation, so as to ensure the fairness of online evaluation.

User Study

For evaluation metrics that are not able to calculate, user study is a better method. By recruiting users to use the recommender system and obtain their real experience, user study can help recommender system adjust and optimize for real use scenarios. When doing user studies, researchers need to ensure that the population distribution of the users surveyed is consistent with that of the real users. For example, if the recommender system is developed for all age groups, then the users surveyed should also be recruited from all age groups. Meanwhile, in the process of user testing, researchers should try their best to guide and help users complete the system testing to improve user experience and satisfaction. The advantage of user study is that compared with offline evaluation and online evaluation, it can obtain users' experience in real scenarios, which is of great significance to the adjustment and optimization of recommender system. However, the recruitment of the users surveyed is the biggest difficulty of the user study method because the recruitment of users needs to invest a huge amount of money and manpower. Additionally, it is also hard to ensure that the recruited users are consistent with the real users.

5.2.4 Cold-Start Problem

The cold-start problem in recommender systems means that when a new user or item enters the recommender system, the system cannot quickly recommend the appropriate items to the new user or recommend the new item to the appropriate users. At present, there are many solutions to the cold-start problem. This section briefly introduces four common methods, including popular item recommendation, recommendation with additional information, recommendation with expert annotations, and conversational recommendation.

Popular Item Recommendation

Recommender systems can record some popular and hot items. When a new user comes in, the system can recommend these items to the user. Recommender systems constantly adjust what to recommend according to users' feedback on these items to improve the quality of recommendation and finally realize personalized recommendation of new users. Although this method is easy to implement, for users who prefer niche items, the user experience will be affected because the recommender system cannot obtain the information about user preferences and provide personalized recommendation in a short time.

Recommendation with Additional Information

Recommender systems can request access to the registration information of a new user, such as the user's gender, age, etc., or study the user's preference through questionnaires. For example, a music recommender system can let users choose the types of music or singers they like when they register. When the additional information of the user is obtained, the system can use content-based recommendation algorithms to select items that meet the user's preference and recommend to the user. New users' social information can also help improve the accuracy of recommendation. Recommender systems can request some information in the user's social account, such as friends, groups of interests, to help the systems establish accurate user profiles and realize personalized recommendation in the early stage.

Recommendation with Expert Annotations

Before being added into the recommender system, new items can be annotated by experts to specify some key attributes of the items. For

example, information such as the release time, director, and actors can be manually added for a new movie, and information such as the type, the singer, and the album can be added for a new song. After the attribute annotation of the new item is completed, the recommender system can quickly calculate the audience of the new item and recommend it to the appropriate users. This method can effectively improve the accuracy of new item recommendation, but it needs more manpower.

Conversational Recommendation

Conversational recommendation can understand users' intentions and preferences through dialogues with users and then realize personalized recommendation. For example, after a new user enters the recommender system, the conversational system constantly understands the user's needs based on the user's input and generates new questions in the meantime, to further explore the user's preferences. After collecting enough information, the system can make personalized recommendation.

5.3 Responsible Recommendation

Recommender systems need to interact with users frequently, which includes collecting user data, training recommendation models, and displaying recommendation results. In the process of interaction, how to ensure that the rights of users will not be infringed by the system is crucial to the success of the recommender system. Therefore, this section describes how to technically reduce the potential risk that algorithms pose to users, that is, how to realize responsible recommendation.

5.3.1 User Privacy

If a recommender system wants to obtain highly personalized and accurate recommendation, it must obtain and fully understand users' historical interaction information and real-time demand. The recommendation quality depends on the scale, accuracy, diversity, and timeliness of the data collected by the system. However, the collection, involving a large number of user behavior records and private information, inevitably makes users concern about privacy leakage. In existing recommender systems, this is an inevitable problem of "privacy-personalization trade-off". Therefore, it is a problem worth studying in the current era of big data that how to collect

and mine the value of user data while ensuring that user privacy is not leaked to the recommender system and any third parties.

In 2015, eBay suffered a hacker attack that compromised 145 million user accounts, including user names, addresses, birth information, and passwords. In 2018, Facebook suffered a number of privacy breaches, with the personal information of tens of millions of users, including names and contact details, leaked due to software vulnerabilities and hacker attacks. In 2018, an information disclosure incident also occurred in a Chinese domestic express company, with more than 1 billion pieces of delivery data, including names, mobile phone numbers, and home addresses, being sold online. At the early stage of the research, the privacy of users in recommender systems has been highly valued by researchers. The well-known Netflix Prize recommendation algorithm competition, for example, had brought researchers' enthusiasm for recommendation algorithms to a high point, but it was later forced to stop due to user privacy leakage from the open datasets. The researchers linked the Netflix dataset with the IMDb dataset and discovered the political preferences and sensitive information of some users [36]. In addition to these external factors, some internal factors may also lead to user privacy infringement. For example, driven by commercial interests, service providers may violate the privacy provisions by accessing or collecting user data without authorization and sharing data with third parties. At the same time, employees in the enterprise may use their access rights to monitor users' privacy for some interests or other reasons.

With the occurrence of such incidents and the enhancement of people's awareness of privacy protection, users are paying more and more attention to protect their data privacy and avoid their private information being collected by Internet applications. Governments are also aware of the importance of data privacy and have issued laws and regulations on data privacy and privacy protection. In recent years, China has issued the Measures for Data Security Management (Draft for Comments) and the Data Security Law of the People's Republic of China. The EU has issued the General Data Protection Regulation (GDPR). California in the USA has issued the California Consumer Privacy Act (CCPA). The promulgation of these regulations guarantees the privacy of user data to some extent, and commercial organizations cannot collect user data without supervision as before any more.

In addition to laws and regulations, researchers can also improve the original algorithms and design a more reasonable recommender system architecture to protect user privacy. These methods fall into 3 main categories: architecture-based methods, algorithm-based methods, and federated-learning-based methods.

The architecture-based methods aim at minimizing the threat of data leakage. For example, distributed storage of user data can effectively reduce the damage caused by the disclosure of a single data source, and distributed recommendation processes can increase the difficulty of unauthorized access to data. Heitmann et al. [22] proposed a candidate architecture in which users host profile data and decide which parts of the data can be accessed by which service providers. Only with a specific certificate can an application access the corresponding part of the profile data through the API and use it for recommendation. Based on the idea of distributed recommendation, Hecht et al. [20] proposed a recommendation process based on P2P (peer-to-peer) system, comparing the data similarity between the user and the others to obtain possible recommendation results. This eliminates the need for a central server to store personal data centrally. However, these methods can still expose user data to other users and require high computing power on local devices.

The algorithm-based methods modify the original data so that after modification user privacy will not be compromised even if the data or model outputs are obtained by a third party. These methods consist mainly of data perturbation and homomorphic encryption algorithms. The methods based on data perturbation design effective data perturbation techniques to protect user privacy, such as adding noises from zero-mean Gaussian distribution to user ratings and then sending the perturbed data to the server to realize privacy protection. Agrawal et al. [2] introduced additive perturbation technique into the field of data mining for the first time to protect data privacy by adding Gaussian noise to the original value. In 2009, the concept of differential privacy was introduced into the field of recommender systems for the first time by McSherry et al. [35] from Microsoft Research, providing user privacy protection with theoretical guarantee. Differential privacy is also one of the most important methods in data perturbation, which can reduce the risk of user privacy disclosure by disturbing inputs or outputs of recommender systems. In 2015, Berlioz et al. [5] evaluated the trade-off between privacy effectiveness and

recommendation accuracy when applying differential privacy to matrix factorization. The solutions based on encryption can reduce the user privacy leakage problem more strictly. At present, homomorphic encryption is the main method. Its main characteristic is that the calculation can be performed on the encrypted data, and the decrypted results are the same as those based on plain text. In 2002, Canny [7] proposed a matrix factorization framework based on homomorphic encryption. Users encrypt local data through public keys and share private keys through distributed key sharing. More than half of the users complete data decryption through voting. This means that at least half of the users should be online at the same time to complete model training and the calculation of recommendation results. The main disadvantage of this method includes high computation time, storage space, and communication cost, so it is only suitable for small -scale recommender systems.

Federated learning is a machine learning framework for privacy protection proposed by Google in 2016, which can realize model update without collecting user data. It was first used for model update on mobile phones and later extended to more application scenarios. Traditional machine learning methods require training data to be centralized in a single machine or data center. However, federated learning completes the machine learning model training through the distributed collaboration of thousands of users. In the training process, all user data are only saved in users' own devices, and only the intermediate calculation results are shared with each other instead of the original data, so as to protect user privacy. Federated learning avoids the privacy risks and data security problems caused by centralized data collection and storage and can use all users' data to train the model. In fact, long before the concept of federated learning was proposed, the same idea had been used to protect user privacy in recommendation algorithms. For example, in a paper published in 2016, Dongsheng Li et al. [27] proposed to calculate the similarity between items based on the sharing of intermediate results among users, which can strictly protect users' privacy during the calculation process. In contract, the federated learning framework is more general, where a host (a central server or a single member) initiates the learning task. Under the coordination of the host, every member trains the model based on local data. The host then collects the training results of all members and safely aggregates them into a global model and shares the updated global model

with each participant. The process is repeated until the global model achieves the training objective, such as convergence. Finally, all participants share the globally optimal machine learning model. Throughout the process, the original data of the participants are kept locally and will not be exchanged or transferred. Federated learning allows for some deviation in the accuracy of the training model but can provide data security and privacy protection for all participants. In recent years, many researchers are devoted to the realization of machine learning models based on federated learning, such as deep neural networks based on federated learning. Additionally, researchers also try to integrate different privacy protection methods, such as combining differential privacy, homomorphic encryption, and other theoretically rigorous methods with federated learning, which provides a theoretical basis for introducing federated learning into recommender systems.

As people pay more attention to privacy protection, more and more privacy protection algorithms will be applied to recommender systems. Additionally, with the development of other basic information technologies, such as mobile computing and 5G, many performance bottlenecks of privacy-preserving recommendation algorithms will be alleviated, and privacy-preserving recommendation algorithms will be developed further.

5.3.2 Explainability

In addition to providing recommendation results, explainable recommendation also requires explanation for the recommendation, which can improve the transparency, persuasiveness, effectiveness, credibility, and user satisfaction of the recommender system. Additionally, the development of explainable recommendation can help system designers diagnose, debug, and improve the recommendation algorithm. Explainable recommendation mainly includes explanation generation and human–computer interaction. The former focuses on how to generate explanations, while the latter focuses on how to present the explanations of recommendation results. This section will briefly introduce the relevant knowledge of explainable recommendation from these two aspects.

Technical Routes to Recommendation Generation

According to the position of the generation of recommendation explanation in the whole recommendation pipeline, explainable recommendation can be

divided into two technical routes: explainable recommendation models and explaining after recommendation. Some basic ideas and representative methods of each technical route are introduced below.

Explainable recommendation models are expected to have certain transparency, so that the explanation of corresponding recommendation results can be generated at the same time when generating recommendation. At present, the main techniques applied in recommender systems can be modified to obtain model explainability. Table 5.1 summarizes the improvement schemes for explainable recommendation based on different technical routes and their corresponding representative models. This table is a partial summary. Explainable recommendation is a rapidly developing field. In addition to the technical routes and improvement schemes mentioned in the table, there are many outstanding techniques that have not been covered.

Table 5.1 Explainable recommendation improvement schemes based on different technical routes and their corresponding representative Models

	Explainable improvement	
Technical route	**scheme**	**Representative model**
Factorization models	Align with explicit features	Explicit factor models (EFMs) [49]
	Neighbor-style explanations	Fast influence analysis (FIA) [13]
	Relevant users/items explanations	Explainable matrix factorization (EMF) [1]
	Explanations based on features extracted from reviews	Sentiment utility logistic model (SULM) [4]
	Build topic models based on reviews	Hidden factor and topic (HFT) [34]
	Integrate other structural data	The FacT [40]
Graph models (exclusive of graph neural networks)	Graph propagation	TriRank [19]
	Graph clustering	Overlapping co-CLuster Recommendation (OCuLaR) [21]
Deep learning models	Apply attention mechanism on review data	Dual Attention-Based Model (D-Attn) [37] Deep Explicit Attentive Multi-view Learning Model (DEAML) [16]

	Explainable improvement	
Technical route	**scheme**	**Representative model**
		Neural Attentional Regression model with Review-level Explanations (NARRE) [9]
	Auto-generated text explanations based on natural language generation	Automatic Generation of Natural Language Explanations [15]
	Crowd-based text explanations	Crowd-Based Personalized Natural Language Explanations for Recommendations [8]
	Visual explanations	Visually Explainable Collaborative Filtering (VECF) [11]
	Explanations based on capsule network logit units	Capsule Network-Based Model for Rating Prediction with User Reviews (CARP) [26]
	Explanations based on user historical behavior impacts	Sequential Recommendation with User Memory Networks (RUM) [12]
Knowledge graph models	Entity-based explanations	Programming with Personalized Page Rank (ProPPR) [46]
	Path-based explanations	Policy-Guided Path Reasoning (PGPR) [3]
	Graph propagation	RippleNet [44]
	Integrate induction of rules from knowledge graph with construction of rule-guided neural recommendation model	Jointly Learning Explainable Rules for Recommendation with Knowledge Graph [33]

Different from the above explainable recommendation models, explaining after recommendation is to build a separate model to explain the recommendation results from the recommendation model. When the recommender system architecture is complex, it is difficult to provide embedded explainability in the model, but it is often not that hard to provide a user understandable explanation for the recommendation results. For example, an e-commerce platform recommends products to users through a complex hybrid recommender system and gives recommendation explanations such as “80% of your friends have bought the product” based on simple statistics. It is worth noting that “finding” an explanation for the recommendation result is not providing a false explanation. In machine learning, a prominent idea of model agnostic explanation is to approximate

complex models with simple models because such simple models help to understand parts of complex models.

From the perspective of cognitive science, constructing explainable recommendation model and explaining after recommendation correspond to the two ways in which humans provide explanations for behaviors. The former can be analogous to asking a person who is used to thinking twice before he/she acts to explain his/her actions. He/she can analyze in great detail the reasons for making the decision. But in real life, there are also many cases where you make a decision based on intuition first and then try to explain that decision. These cases correspond to the latter. No one way is better than the other. It needs to consider the application requirements of the recommender system when deciding how to choose.

Presentation of Explainable Recommendation

According to different technical routes of recommendation generation, there are various ways to present recommendation explanation, including the following four: user/item-based explanation, feature-based explanation, text-based explanation, and visual-based explanation.

User/item-based explanations are common in collaborative filtering recommender systems, where relevant users/items are provided as explanations, such as “80% of users similar to you like item A”. It is worth noting that in recommender systems based on collaborative filtering, relevant users are generally defined as users with similar behavior patterns. Under this definition, a user may not know the other relevant users, which makes the explanation less effective. Additionally, this setting may also lead to user privacy issues. Social recommendation is a solution to this problem. Compared with exposing one’s preferences to strangers, sharing preferences among friends will be more acceptable in terms of privacy and explanation.

Feature-based explanation is closely related to content-based recommender systems. In content-based recommendation, the system provides recommendation by matching user features with the features of candidate items, so intuitive explanations can be made based on the features. One common approach is to present the features that match the user, such as the illustration of car recommendation shown in Fig. 5.1. Compared with the rich features of items, users’ features are usually relatively simple, among which the demographic information and user reviews are common in recommender systems. User’s demographic

information can be used to generate demographic-based explanations, such as “80% of users of your age like item A”. Opinion-based explanation is an important way to make use of user reviews for recommendation explanation. This method extracts triplets of the user’s “Aspect-Opinion-Sentiment” as features from the user’s reviews (or other data sources) and provides explanations by showing how user preferences match the items in all aspects.

Fig. 5.1 Illustration of explanation for car recommendation

As the name implies, text-based explanation provides recommendation explanation in the form of plain text, which can be divided into template-based text explanation and generative text explanation in terms of flexibility. Template-based text explanation first creates an explanation template and then fills the template with different words to provide personalized explanations for different users. For example, Zhang et al. [49] proposed that feature-based templates could be designed to explain why “recommend” and “disrecommend” decisions are made for users. Item features are variables in the template and are filled adaptively with those that are considered as the most relevant by the recommendation model. The template designed by Wang et al. [45] contains both features and opinionated phrases, providing more opinionated textual explanations through various combinations of features and opinionated phrases. Although researchers are constantly improving the complexity of templates, template-based explanation still suffers from lack of diversity and personalization. Generative text explanation can solve the explanation

personalization problem by using the technologies such as natural language generation to generate recommendation explanation directly. Li et al. [28] used recurrent neural networks to automatically generate recommendation explanations simulating those on Amazon and Yelp. Lu et al. [32] proposed a method of jointly learned recommendation and explanation, which uses the sequence-to-sequence architecture commonly used in the field of natural language processing to generate text explanation. Compared with template-based text explanation, generative text explanation has higher degree of freedom, but also faces more complicated noise problems, such as whether the explanation sentences are smooth and whether the semantics are consistent. Therefore, the current recommender systems favor explanation methods based on templates.

Visual-based explanations are presented by highlighting or boxing in the picture the parts corresponding to the main features of the generated recommendation. As shown in Fig. 5.2, the model can capture the most important areas in the image as explanations manually or by attention mechanism. Lin et al. [30] studied the explainability problem in outfit recommendation, proposed a method with mutual attention mechanism to correlate features in images with features in texts, and then used important text fragments and image regions as the recommendation explanation. Chen et al. [11] proposed a region highlighting method to select explanation regions from images. Since different users have different visual preferences, this method associates users' preferences with image regions through the attention mechanism to provide explainable recommendation. The research on visually explainable recommendation is still in its early stage and there are few related studies. As deep learning-based image processing technology continues to advance, images will be integrated into recommender systems to obtain better accuracy, efficiency, and explainability.

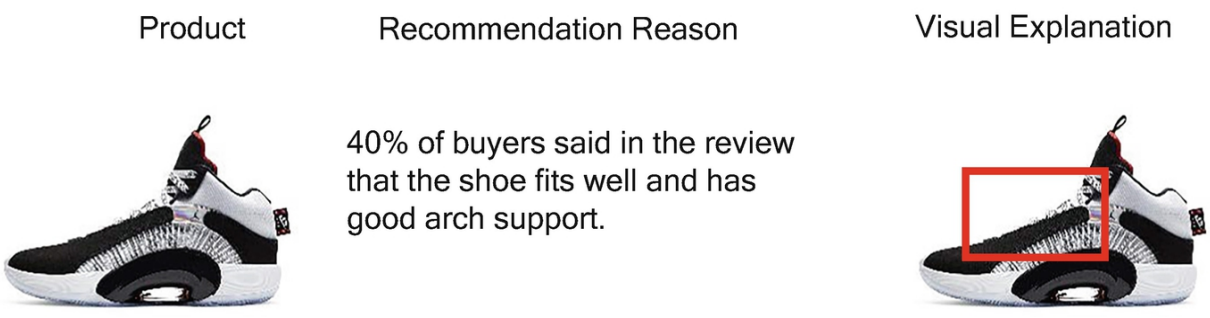


Fig. 5.2 Visual explanation of product recommendation

5.3.3 Algorithm Bias

In recent years, social fairness has attracted wide attention from all walks of life, and the algorithm bias has also become a hot topic in the field of machine learning. In the field of recommender systems, the algorithm bias that researchers pay attention to mainly includes: feature bias or popularity bias, statistical fairness, conformity bias, and long-term fairness, etc.

Feature bias mainly focuses on the unfairness of supervised learning relying too much on predetermined sensitive features. For example, recommender systems may amplify the association between "gender" and "film genre", so that adventure or horror films favored by male users may not be recommended to female users or the possibility of recommendation is very low, leading to the deviation between the recommendation results and the actual situation, and bringing gender injustice. If popularity is regarded as a feature of an item, the popularity bias can also be considered as a special case of feature bias. Popularity bias means that the recommender system tends to recommend popular items to users, so unpopular items will be treated unfairly by the recommender system. For high-quality new items, this popularity bias will bring serious problems. For example, some high-quality films are rarely recommended to users due to their small audience or low early ratings. One way to solve feature bias is to ensure the independence between recommendation results and features, so that the recommendation scores satisfy the following formula [23]:

$$\Pr(R|V) = \Pr(R), \tag{5.1}$$

where R denotes the rating, and V represents a given feature, such as popularity, gender, occupation, etc. The above formula indicates that the recommendation rating should be independent of the features in V, that is, to ensure that the rating is unbiased. In order to realize the above independence in the optimization process, a penalty term can be added to the optimization objective, that is, to realize the independence of R and V by reducing the mutual information of R and V.

Statistical fairness mainly focuses on whether the recommendation results, obtained by one user or a group of users, are consistent with the distribution of user interests. One way to measure statistical fairness is Demographic Parity, that is, whether the distribution of recommendation

results is the same as the distribution of user interest or the distribution of users' interests within a group. For example, if 70% of the users in a group like comedy movies, 30% like action movies, then the recommendation results should have 70% comedy movies and 30% action movies. For individual users, one way to achieve statistical fairness is recommendation calibration [38], that is, to design metrics to measure the statistical fairness of the recommendation results, such as the KL divergence between the recommendation distribution and the user interest distribution, and then reorder the recommendation results according to the metrics. For group users, one way to achieve statistical fairness is to ensure that the recommendation bias is no more than the bias of the input data [42], that is, to not amplify the statistical bias. For example, the probability that an item is recommended in each group should be the same as the probability that the item appears in the training data. If there is a deviation between the two, it can be solved by reordering.

Conformity bias means that users are easily influenced by other users' opinions and tend to give up their unique interests in order to be in line with the opinions of the majority, such as the herding effect. Lederrey et al. [24] did a comparative analysis of two beer rating websites and found that when the initial ratings of the same beer are quite different, their final ratings will also have a huge difference. That is, when a beer got a lot of bad reviews initially, it will have a lower overall rating on the website. On the contrary, if a beer received more positive initial reviews, that beer will have a higher overall rating on the website. In order to solve this problem, Liu et al. [31] proposed a matrix factorization-based conformity modeling technique, which changes the preference-based recommendation in matrix factorization to recommendation that comprehensively considers user preferences and public opinions. In addition to modeling conformity directly, related research also considers the impact of social relationship on user conformity and proposes methods to eliminate social conformity from user prediction ratings. For example, Wang et al. [47] pointed out that there are strong social ties and weak social ties in social networks, and the influence of different social ties on users' interests is also different. Therefore, they proposed to learn the different social ties of each user, put the differences in strength into the modeling of recommendation algorithm, and control the influence of social relations on users' ratings through hyperparameters.

The above issues of fairness only consider short-term or static fairness, while the interaction between the recommender system and users is long-term and dynamic. Therefore, it is necessary to take into account how to ensure the fairness in recommendation in a long-term and dynamic environment, namely, long-term fairness. For popularity bias, which is one of the issues of fairness in recommender systems, the popularity of the item changes over time, so the strategy to ensure the fairness in recommendation needs to track this dynamic bias and applies corresponding correction or calibration. One way to improve long-term fairness is to adopt the idea of reinforcement learning and define long-term fairness problem as a Constrained Markov Decision Process [17]. The model can adjust the recommendation strategy according to the dynamic bias to ensure that the demand for fairness can be continuously satisfied. Although the above method has a strict guarantee on fairness, it is complex in application. Therefore, a simpler method incorporating randomness can be used to improve the long-term fairness in recommendation. Borges et al. [6] found that adding a simple random noise component to the sampling process of variational autoencoder can improve the long-term fairness of the model, but this method would compromise the accuracy of the recommendation results. Experiments show that this method can reduce the algorithm bias by 76% despite a decrease of 5% in the recommendation accuracy.

5.4 Summary

This chapter introduces the hotspots of recommender system research, the key challenges of recommender system application, and how to achieve responsible recommendation technically. These contents may become the key of recommender system research and application in the future, so they need the continuous attention of researchers and developers.

References

1. Behnoush Abdollahi and Olfa Nasraoui. “Explainable Matrix Factorization for Collaborative Filtering”. In: *Proceedings of the 25th International Conference Companion on World Wide Web*. WWW ’16 Companion. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 5–6. ISBN: 9781450341448. https://doi.org/10.1145/2872518.2889405.

2. Rakesh Agrawal and Ramakrishnan Srikant. “Privacy-Preserving Data Mining”. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 439–450. ISBN: 1581132174. https://doi.org/10.1145/342009.335438.
3. Qingyao Ai et al. “Learning Heterogeneous Knowledge Base Embeddings for Explainable Recommendation”. In: *Algorithms* 11.9 (2018), p. 137. https://doi.org/10.3390/a11090137.
4. Konstantin Bauman, Bing Liu, and Alexander Tuzhilin. “Aspect Based Recommendations: Recommending Items with the Most Valuable Aspects Based on User Reviews”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 717–725. ISBN: 9781450348874. https://doi.org/10.1145/3097983.3098170.
5. Arnaud Berlioz et al. “Applying Differential Privacy to Matrix Factorization”. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. RecSys ’15. Vienna, Austria: Association for Computing Machinery, 2015, pp. 107–114. ISBN: 9781450336925. https://doi.org/10.1145/2792838.2800173.
6. Rodrigo Borges and Kostas Stefanidis. “Enhancing Long Term Fairness in Recommendations with Variational Autoencoders”. In: *Proceedings of the 11th International Conference on Management of Digital EcoSystems*. MEDES ’19. Limassol, Cyprus: Association for Computing Machinery, 2020, pp. 95–102. ISBN: 9781450362382. https://doi.org/10.1145/3297662.3365798.
7. J. Canny. “Collaborative filtering with privacy”. In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. 2002, pp. 45–57. https://doi.org/10.1109/SECPRI.2002.1004361.
8. Shuo Chang, F. Maxwell Harper, and Loren Gilbert Terveen. “Crowd-Based Personalized Natural Language Explanations for Recommendations”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys ’16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 175–182. ISBN: 9781450340359. https://doi.org/10.1145/2959100.2959153.
9. Chong Chen et al. “Neural Attentional Rating Regression with Review-Level Explanations”. In: *Proceedings of the 2018 World Wide Web Conference*. WWW ’18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1583–1592. ISBN: 9781450356398. https://doi.org/10.1145/3178876.3186070.
10. Qibin Chen et al. “Towards Knowledge-Based Recommender Dialog System”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1803–1813. https://doi.org/10.18653/v1/D19-1189. URL: https://aclanthology.org/D19-1189.
11. Xu Chen et al. “Personalized Fashion Recommendation with Visual Explanations Based on Multimodal Attention Network: Towards Visually Explainable Recommendation”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’19. Paris, France: Association for Computing Machinery, 2019, pp. 765–774. ISBN: 9781450361729. https://doi.org/10.1145/3331184.3331254.

12. Xu Chen et al. “Sequential Recommendation with User Memory Networks”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM ’18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 108–116. ISBN: 9781450355810. https://doi.org/10.1145/3159652.3159668.
13. Weiyu Cheng et al. “Incorporating Interpretability into Latent Factor Models via Fast Influence Analysis”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 885–893. ISBN: 9781450362016. https://doi.org/10.1145/3292500.3330857.
14. Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. “Towards Conversational Recommender Systems”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 815–824. ISBN: 9781450342322. https://doi.org/10.1145/2939672.2939746.
15. Felipe Costa et al. “Automatic Generation of Natural Language Explanations”. In: *Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion*. IUI ’18 Companion. Tokyo, Japan: Association for Computing Machinery, 2018. ISBN: 9781450355711. https://doi.org/10.1145/3180308.3180366.
16. Jingyue Gao et al. “Explainable Recommendation through Attentive Multi-view Learning”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27—February 1, 2019*. AAAI Press, 2019, pp. 3622–3629. https://doi.org/10.1609/aaai.v33i01.33013622.
17. Yingqiang Ge et al. “Towards Long-Term Fairness in Recommendation”. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM ’21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 445–453. ISBN: 9781450382977. https://doi.org/10.1145/3437963.3441824.
18. Alexandre Gilotte et al. “Offline A/B Testing for Recommender Systems”. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM ’18. Marina Del Rey, CA, USA: Association for Computing Machinery, 2018, pp. 198–206. ISBN: 9781450355810. https://doi.org/10.1145/3159652.3159687.
19. Xiangnan He et al. “TriRank: Review-Aware Explainable Recommendation by Modeling Aspects”. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM ’15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 1661–1670. ISBN: 9781450337946. https://doi.org/10.1145/2806416.2806504.
20. Fabio V. Hecht et al. “Radiommender: P2P on-line radio with a distributed recommender system”. In: *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. 2012, pp. 73–74. https://doi.org/10.1109/P2P.2012.6335817.
21. Reinhard Heckel et al. “Scalable and Interpretable Product Recommendations via Overlapping Co-clustering”. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 2017, pp. 1033–1044. https://doi.org/10.1109/ICDE.2017.149.

22. Benjamin Heitmann et al. “An Architecture for Privacy-Enabled User Profile Portability on the Web of Data”. In: *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*. HetRec ’10. Barcelona, Spain: Association for Computing Machinery, 2010, pp. 16–23. ISBN: 9781450304078. https://doi.org/10.1145/1869446.1869449.
23. Toshihiro Kamishima et al. “Efficiency Improvement of Neutrality-Enhanced Recommendation”. In: *Proceedings of the 3rd Workshop on Human Decision Making in Recommender Systems in conjunction with the 7th ACM Conference on Recommender Systems (RecSys 2013), Hong Kong, China, October 12, 2013*. Ed. by Li Chen et al. Vol. 1050. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 1–8. URL: http://ceur-ws.org/Vol-1050/paper1.pdf.
24. Gael Lederrey and Robert West. “When Sheep Shop: Measuring Herding Effects in Product Ratings with Natural Experiments”. In: *Proceedings of the 2018 World Wide Web Conference*. WWW ’18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 793–802. ISBN: 9781450356398. https://doi.org/10.1145/3178876.3186160.
25. Wenqiang Lei et al. “Conversational Recommendation: Formulation, Methods, and Evaluation”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 2425–2428. ISBN: 9781450380164. https://doi.org/10.1145/3397271.3401419.
26. Chenliang Li et al. “A Capsule Network for Recommendation and Explaining What You Like and Dislike”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR’19. Paris, France: Association for Computing Machinery, 2019, pp. 275–284. ISBN: 9781450361729. https://doi.org/10.1145/3331184.3331216.
27. Dongsheng Li et al. “An algorithm for efficient privacy-preserving item-based collaborative filtering”. In: *Future Generation Computer Systems* 55 (2016), pp. 311–320. ISSN: 0167-739X. https://doi.org/10.1016/j.future.2014.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0167739X14002374.
28. Piji Li et al. “Neural Rating Regression with Abstractive Tips Generation for Recommendation”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 345–354. ISBN: 9781450350228. https://doi.org/10.1145/3077136.3080822.
29. Raymond Li et al. “Towards Deep Conversational Recommendations”. In: *Advances in Neural Information Processing Systems 31. 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*. Montréal, Canada: Curran Associates Inc., 2018, pp. 9725–9735.
30. Yujie Lin et al. “Explainable Outfit Recommendation with Joint Outfit Matching and Comment Generation”. In: *IEEE Transactions on Knowledge and Data Engineering* 32.8 (2020), pp. 1502–1516. https://doi.org/10.1109/TKDE.2019.2906190.
31. Yiming Liu, Xuezhi Cao, and Yong Yu. “Are You Influenced by Others When Rating? Improve Rating Prediction by Conformity Modeling”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys ’16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 269–272. ISBN: 9781450340359. https://doi.org/10.1145/2959100.2959141.

32. Yichao Lu, Ruihai Dong, and Barry Smyth. “Why I like It: Multi-Task Learning for Recommendation and Explanation”. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys ’18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018, pp. 4–12. ISBN: 9781450359016. https://doi.org/10.1145/3240323.3240365.
33. Weizhi Ma et al. “Jointly Learning Explainable Rules for Recommendation with Knowledge Graph”. In: *The World Wide Web Conference*. WWW ’19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 1210–1221. ISBN: 9781450366748. https://doi.org/10.1145/3308558.3313607.
34. Julian McAuley and Jure Leskovec. “Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text”. In: *Proceedings of the 7th ACM Conference on Recommender Systems*. RecSys ’13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 165–172. ISBN: 9781450324090. https://doi.org/10.1145/2507157.2507163.
35. Frank McSherry and Ilya Mironov. “Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders”. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD ’09. Paris, France: Association for Computing Machinery, 2009, pp. 627–636. ISBN: 9781605584959. https://doi.org/10.1145/1557019.1557090.
36. Arvind Narayanan and Vitaly Shmatikov. “Robust De-anonymization of Large Sparse Datasets”. In: textit2008 IEEE Symposium on Security and Privacy (sp 2008). 2008, pp. 111–125. https://doi.org/10.1109/SP.2008.33.
37. Sungyong Seo et al. “Interpretable Convolutional Neural Networks with Dual Local and Global Attention for Review Rating Prediction”. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. RecSys ’17. Como, Italy: Association for Computing Machinery, 2017, pp. 297–305. ISBN: 9781450346528. https://doi.org/10.1145/3109859.3109890.
38. Harald Steck. “Calibrated Recommendations”. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys ’18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018, pp. 154–162. ISBN: 9781450359016. https://doi.org/10.1145/3240323.3240372.
39. Yueming Sun and Yi Zhang. “Conversational Recommender System”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR ’18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, pp. 235–244. ISBN: 9781450356572. URL: https://doi.org/10.1145/3209978.3210002.
40. Yiyi Tao et al. “The FacT: Taming Latent Factor Models for Explainability with Factorization Trees”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’19. Paris, France: Association for Computing Machinery, 2019, pp. 295–304. ISBN: 9781450361729. https://doi.org/10.1145/3331184.3331244.
41. Yi-Ting Tsai et al. “A Cross-Domain Recommender System Based on Common-Sense Knowledge Bases”. In: *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. 2017, pp. 80–83. https://doi.org/10.1109/TAAI.2017.48.

42. Virginia Tsintzou, Evaggelia Pitoura, and Panayiotis Tsaparas. *Bias Disparity in Recommendation Systems*. 2018. https://doi.org/10.48550/ARXIV.1811.01461. URL: https://arxiv.org/abs/1811.01461.

43. Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30. 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, California, USA: Curran Associates, Inc., 2017, pp. 6000–6010. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

44. Hongwei Wang et al. “RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems”. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM ’18. Torino, Italy: Association for Computing Machinery, 2018, pp. 417–426. ISBN: 9781450360142. URL: https://doi.org/10.1145/3269206.3271739.

45. Nan Wang et al. “Explainable Recommendation via Multi-Task Learning in Opinionated Text Data”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR ’18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, pp. 165–174. ISBN: 9781450356572. https://doi.org/10.1145/3209978.3210010.

46. William Yang Wang, Kathryn Mazaitis, and William W. Cohen. “Programming with Personalized PageRank: A Locally Groundable First-Order Probabilistic Logic”. In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM ’13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 2129–2138. ISBN: 9781450322638. https://doi.org/10.1145/2505515.2505573.

47. Xin Wang et al. “Learning Personalized Preference of Strong and Weak Ties for Social Recommendation”. In: *Proceedings of the 26th International Conference on World Wide Web*. WWW ’17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 1601–1610. ISBN: 9781450349130. https://doi.org/10.1145/3038912.3052556.

48. Yixin Wang et al. “Causal Inference for Recommender Systems”. In: *Proceedings of the 14th ACM Conference on Recommender Systems*. RecSys ’20. Virtual Event, Brazil: Association for Computing Machinery, 2020, pp. 426–431. ISBN: 9781450375832. https://doi.org/10.1145/3383313.3412225.

49. Yongfeng Zhang et al. “Explicit Factor Models for Explainable Recommendation Based on Phrase-Level Sentiment Analysis”. In: *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR ’14. Gold Coast, Queensland, Australia: Association for Computing Machinery, 2014, pp. 83–92. ISBN: 9781450322577. https://doi.org/10.1145/2600428.2609579.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_6

6. Practical Recommender System

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter focuses on some problems and considerations in industry applications of recommender systems, and discusses the details of these actual applications based on the code in the Microsoft Recommenders repository and a cloud-based reference architecture. Readers are encouraged to follow the steps and methods in the text in a hands-on manner and experiment using the algorithms introduced in previous chapters.

Keywords Industry applications – Microsoft recommenders – Hands-on – Cloud-based architecture

6.1 Introduction

In the previous chapters, the recommender system is introduced and discussed theoretically with details. For the recommender system that is industrially applicable, there are usually key points in addition to recommender system algorithms. Examples of such key points are data management and governance, evaluation metrics, system DevOps, administration, etc. Due to the variety and complexity of the technical details, establishing an industry-grade recommender system is more complicated than the commonly seen software system. In this chapter, the practice of building applicable recommender system is discussed. In combination with the theoretical introduction in the previous chapters, a systematic overview on how to build an industrially applicable recommender system is presented. In addition, Microsoft Recommenders, an open source repository of best practices on recommender systems on GitHub, is used to highlight implementation details for each topic. An end-to-end recommender system is used as an illustration to demonstrate the full lifecycle of building and deploying a recommender system. The hands-on code examples are provided for the readers to practise.

6.2 Architecture and Implementation of Industry-Grade Recommender System

6.2.1 Characteristics of Industry-Grade Recommender System

Nowadays, with the proliferation and rapid growth of Internet technology, more and more corporations are starting to invest in mining the value from their data and information flows to boost revenue growth. Corporations are moving beyond initial adoption of recommender systems for targeted sections of the business and have begun applying this technology across many core areas within the organization. Examples can be found in Alibaba's e-commerce service, TikTok video app, etc. In addition, other than the Internet industry there are effective application of the recommender system technology as well. It is observed that in different industry verticals wherever there are enterprise-client interactions, by nature of the recommender methodology, the uplift of business growth by leveraging the technology is witnessed thanks to its positive impact on building the channel for information exchange.

Due to the differences in various industry verticals, the applicable recommender system in those industries varies. In general, an industry-grade recommender system should be scalable, explainable, maintainable, and configurable. The uniqueness of the recommender system technology lies in the fact that, deploying a

recommender system should meet the specifications from various aspects so as to serve to the sophisticated business request. Examples of such specifications are listed but are not limited to the items below.

In the first place, the mainstream recommender system is mostly based on the machine learning algorithms. Model training and retraining cycle is vital to the business needs. Usually, the choice of the training cycle is dependent on the actual demand. For example, for the use case of e-commerce where the user base size is magnificent, the retraining cycle of the recommender system model would be as short as possible – this is for the reason that the model is capable of capturing the massive feedback from users in real time. The ask for frequent retraining leads to the requirement of performing large-scale machine learning algorithm training with high efficiency, and hence, this becomes one of the critical tasks in designing and developing recommender system. In some special circumstances, due to the dynamics of training data, online learning or reinforcement learning is used to replace the conventional machine learning for an enhancement of recommendation quality.

Usually the feature vectors that represent characteristics of user and product for building a model in a recommender system have high dimensionalities. These high-dimensional representation of data guarantees the generality of the trained model on the data that are not in the training set. It is a challenge to train a model with high-dimensional data – the challenge is even bigger if the recommendation model is a deep learning based one. For the deep learning based model, parameter optimization is pivotal. In some of the deep learning tasks such as natural language processing, the parameter optimization needs to be performed for only once. The cost of using and retraining a pre-trained model is not costly even if the tasks require human involvement. As for the recommender system, however, due to the high velocity and nuanced business relevancy, the parameter optimization needs to be conducted more iteratively compared to the natural language process model building. Also, the model building of a recommender system requires system-level support such that the parameter optimization can be greatly automated for efficiency.

In addition, the algorithms used in recommender systems diversify. Correspondingly, the implementations of these recommender algorithms also vary. Hence, the support from an architectural perspective is needed. In the industrial recommender system, there are usually more than one recommender models deployed such that they collaboratively optimize the final objective. This requires the heterogeneous computing architecture to allow the workloads of different algorithms. For example, a deep learning model may require the distributed GPU cluster for model training whilst the Spark-based model may require the distributed CPU cluster for model training.

Evaluation is often neglected but it plays a vital role. In many recommendation scenarios, the online evaluation metrics are often more important than the offline ones. The offline metrics serve to model selection and parameter optimization whilst the online metrics help evaluate the final gain of the recommender system for business application. In many times the output of the online evaluation and the offline evaluation may not align. This discrepancy requires the human involvement from the engineering side and the business side to jointly choose the appropriate evaluation metrics for both the online and the offline cases that yield the optimal outcome.

And last but not least, a recommender system does not merely rely on the machine learning algorithms, heuristics that originate from business logic sometimes help significantly. This requires the algorithmic engineers who design and implement the recommender systems to take a close eye on the business demand. For those demands that cannot be conveniently abstracted to algorithmic implementation, business-based rules or such can be added to help converge the recommender system to get the optimal gain. These rules may be diversified according to the various needs on the business side, but the general result they help generate may be phenomenal.

6.2.2 Commonly Used Architecture of Recommender System

Technically, a recommender system can be categorized by its deployment characteristics, whether it is an offline recommender system or a real-time recommender system.

6.2.3 Offline Recommender System

As shown in Fig. 6.1, an offline recommender system consists of the following major components, *raw data*, *data preprocessing and feature engineering*, *model training*, and *recommendation and front-end service*.

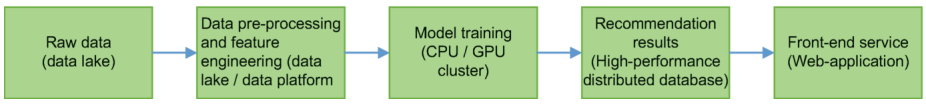


Fig. 6.1 Offline recommender system

Raw data is composed of the data that is required by building a recommender system. Examples of raw data are user behavior, item-related data, log information, text, images, etc. Usually, the data size is rather big (TB or

even PB); therefore, saving the data requires technologies such as data lake so that the data of different formats can be persistent in a centralized way.

Before it is applied for model building, the raw data needs to be preprocessed and featurized. In the recommender system pipeline, these data preprocessing steps use not only the conventional techniques in machine learning such as normalization, standardization, duplication, removal of null value, etc. but also the feature transformation and selection techniques that are unique to recommendation problems. For example, for some recommender algorithms (e.g., Wide and Deep), feature product is used for building model. As a result, the feature products can be generated in the feature engineering steps and then get used in the subsequent model building process. Some deep learning recommender algorithms require embeddings to represent user/item features or the unstructured data (e.g., text, image, etc.). These embedding vectors can be pre-computed in the feature engineering process. Usually, due to the large-scale computation in data preparation, this stage is performed on a big-data platform such as Spark, Kafka, etc.

Model building is the core of the recommender system. Due to the variety of the recommender algorithms, the platform and framework used for model building vary. As aforementioned, the computing platform for model building is usually heterogeneous. For the offline recommender system, the recommendation results can be generated in an offline manner. The collaborative filtering algorithm can be used for offline recommendation. The results of the recommendation pipeline can be generated immediately once the model has been trained.

One of the uniquenesses of the offline recommender system is that, the recommendation results generated from the model can be persisted in the data storage medium. This mechanism guarantees that the computational efforts happening in data preprocessing and model training do not infer the last stage where the front-end service gets recommendation results so that there is no unnecessary latency. This is not to say that there is zero latency in the entire system. Instead, the latency of an offline recommender system is dependent on the results-fetching operation from the front-end service. Therefore, the actual latency of an offline recommender system relies on the data storage for recommendation results persistence. A common data storage medium used in the recommender system is a concurrent and high-performance distributed database. This makes sure that the pre-cached recommendation results in the storage can be obtained consistently and coherently from any circumstances with latency that meets the engineering specifications.

The front-end service is usually a hosted service that is exposed to either a web or a mobile app client to allow the users to access to the recommender system. In the offline recommender system, the dataflow underlying the hood of the user access is that, based on the user-defined information it goes to the database to fetch the pre-computed recommendation results. The industrial implementations of such front-end service are usually based on Docker containers or Kubernetes.

In general, offline recommender system is easy to build and maintain. It does not have high requirement for the latency in data preprocessing and model building. It needs a high-performance underlying infrastructure, especially the database that preserves the pre-computed recommendation results. The offline recommender system is usually applicable in the use cases where the recommendation tasks are highly scalable and the requirement for real-time recommendation is low. Sometimes, the offline recommendation system architecture is also used as a *recall* layer in a complex recommender system to generate the candidate set that can be used for re-ranking in the next stage. This will be discussed with details in the following content.

Real-Time Recommender System

As name suggests, a real-time recommender system generates recommendation results in a real-time manner. Figure 6.2 demonstrates the architecture of a real-time recommender system. Similar to an offline recommender system, a real-time recommender system needs to preprocess the raw data before the data can be used for model building. Differently, the real-time recommender system does not preserve the recommendation results immediately after the model is built. Instead, the model object will be deployed and hosted onto a service such that the front-end queries are sent back to the model for scoring in real time. In Fig. 6.2, in addition to the dataflow from the model training side to the front-end, there is another flow from the data preprocessing module to the front-end which triggers the model scoring and ranking. In the scoring and ranking procedure, the features that have been used for model building are used for model scoring, too, to make sure the feature consistency.

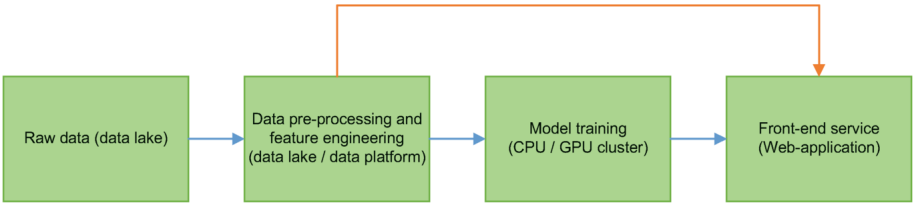


Fig. 6.2 Real-time recommender system

Compared to the offline recommender system, the real-time recommender system is beneficial in capturing the velocity and timely characteristics of the data. Meanwhile, the real-time recommender system is capable of generating recommendation results with larger varieties so that the recommendation performance is enhanced. However, due to the complexity, implementation efforts of a real-time recommender system are also higher than an offline one. The origin of the efforts is mainly model scoring and ranking which needs to be performed in real time. Both scoring and ranking have relatively high complexity, especially for those large-scale recommendation tasks, the model scoring should be particularly optimized to make sure the entire recommendation pipeline meets the latency requirement. Therefore, the front-end of a real-time recommender system usually uses a high-performance framework.

Nowadays, the recommender system implementation does not rely on merely one of the two recommender system architectures. A smarter way of building an applicable recommender system is to conduct a hybrid approach, to allow the two architectures to contribute in various use case scenarios. The benefits of doing this is that, the actual implementation of the recommender system can be achieved according to the feedback mechanism of the users at the front-end. On top of the two fundamental recommender system architecture, there are a lot more components developed in the modern recommender system to optimize and uplift the recommendation performance.

6.2.4 Industrial Implementation of Recommender System

Compared to other types of the artificial intelligence technologies, recommender system has a closer binding with its business application. In the recent years, in different industry verticals, recommender systems that aim at different problems sprout vastly. In this subsection, some of the representative recommender system in industry are briefly introduced.

Amazon is considered to be one of the first companies that put the recommender system technology into application for its core business. To effectively promote its products and enhance user loyalty, Amazon developed the product-based collaborative filtering recommender system. The recommender system calculates the similarity of the products and find those products that have been purchased by customers in the past, based on which the new products are recommended. This classic method looks naive today but then it generated significant impact – at the scale of about 30 million users and millions of products, the product-based recommender system guarantees the system scalability, and compared to other algorithms at that time it was more impactful and reliable [7].

Netflix started applying recommender system in its media streaming service early. In 2006, Netflix launched the “Netflix Prize” competition, from which two classic recommender system algorithms, matrix decomposition and restricted Boltzmann machine, were proposed and shed light on the following research and development of recommendation algorithms. In the following development, the engineers at Netflix progressed the objective of a recommender system from “scoring” to “ranking” so that the recommendation results are more close to users’ psychological preferences than before. In the meantime, architecturally, Netflix proposed the online and offline separation as well as the scoring and ranking components in recommender systems. As shown in Fig. 6.3, the architecture inspires many following designs which are used as references for sophisticated systems [1].

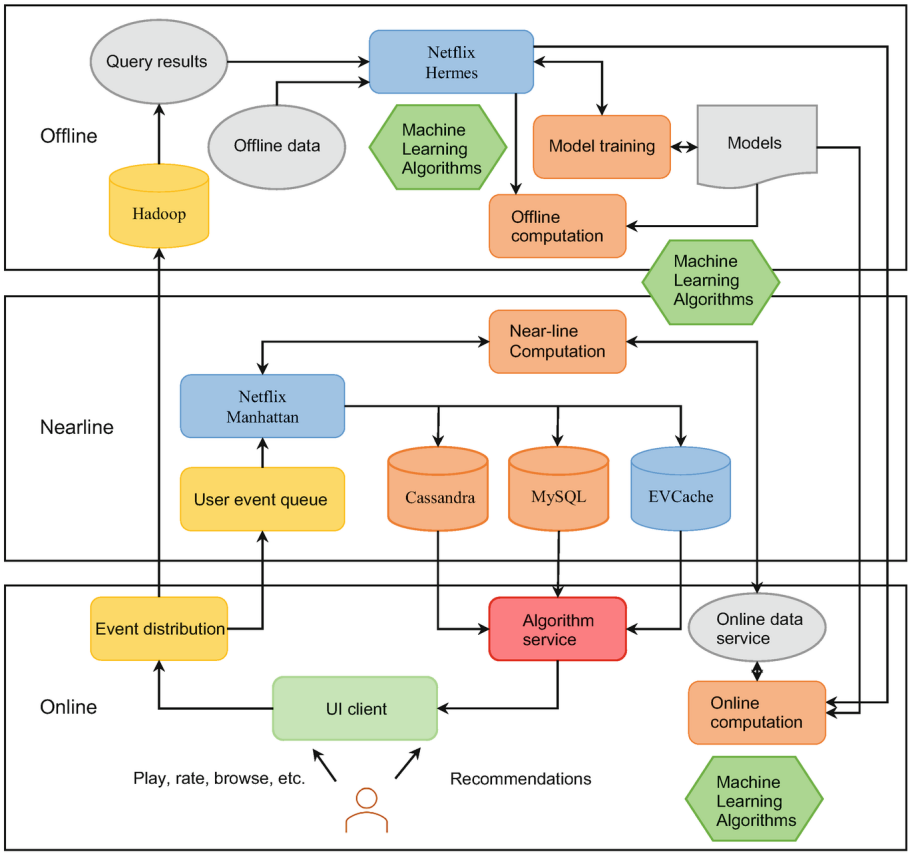


Fig. 6.3 Netflix recommender system architecture

In addition to the aforementioned vertical applications of recommender system, there are also design and implementation of reusable modules for building recommender system at organizations where there is a lack of technological foundation. One of the examples is the Merlin recommender system framework developed by NVIDIA [9]. Merlin is a framework that allows developers to design and implement an end-to-end recommender system that consists of the components of data preprocessing, model building, and model scoring. In Fig. 6.4, compared to the normal recommender system, Merlin is optimized by leveraging the GPU technology to make the model building, especially the deep learning one, more computationally efficient.

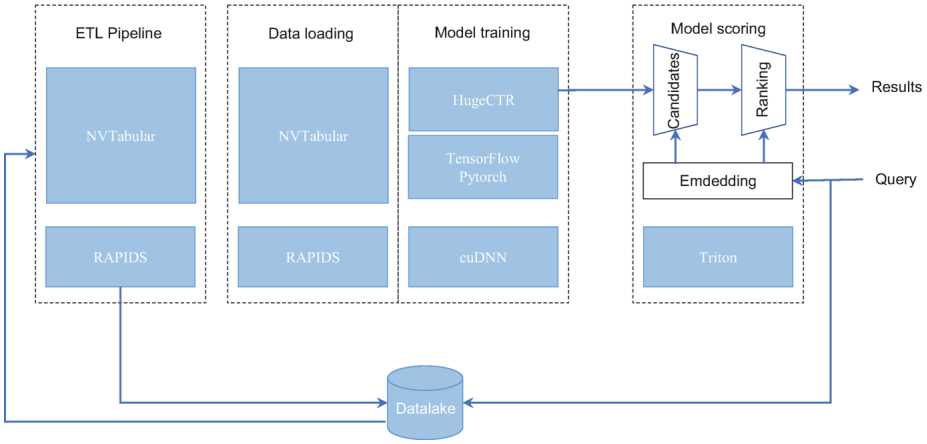


Fig. 6.4 NVIDIA Merlin architecture

6.3 Practices of Recommender System

Microsoft Recommenders collects the classic and advanced methods used in recommender system related topics since its birth. These methods include the key topics such as data preprocessing, algorithm selection, etc. Microsoft Recommenders was developed in the era when the recommender system technology sprouted vastly. Though the research publications in the realm have been produced dramatically, there are pending challenges that apply the recommender system into production. For the organizations that do not have sufficient technological foundation, applying recommender system for its business becomes a task that cannot be easily accomplished. The challenges include but are not limited to the following. In the first place, despite the large volume of publications related to recommender system being generated every year, references that can be used directly for industrial applications are rare. In addition, most of the tools and/or packages that are off-the-shelf in the market for building recommender system are merely aimed at building particular components of a full-fledged system. Also, different packages or tools are implemented differently and independently such that integrating them together is difficult. The last but not the least, majorities of the packages that are developed by researchers or engineers with academic background may not fulfill the requirement of industrial software standard.

Microsoft Recommenders was developed to resolve the aforementioned issues [3]. Microsoft Recommenders was launched and publicized on GitHub as an open sourced project. It uses MIT license. It supports various running environment that uses CPU, GPU, or Spark. It is noted that, as of the year when this book is published, i.e., 2023, Microsoft Recommenders is moved to Linux foundation.

In general, Microsoft Recommenders is featured by the following:

- Microsoft Recommenders provides a set of best practices that can be used for developing and deploying industry-grade recommender system. The best practices cover not merely the classic recommender system technologies but also the advanced ones, which makes Microsoft Recommenders surpass many of the existing packages thanks to the rich implementation references it offers.
- Microsoft Recommenders is designed and developed by following the commonly used design pattern in the modern software development. For example, one of the principles used in development is “evidence-based design.” This principle requires the code implementations in Microsoft Recommenders referenceable, and these implementations have been validated in the real-world applications. These principles help the developers and contributors to commit quality codes to Microsoft Recommenders, which makes them deployable into production environment.
- The persona of Microsoft Recommenders is not the developers in enterprise. Researchers in academia, lecturers in education institute, college school students, etc. can all be the users. Due to the compatibility between the interfaces in Microsoft Recommenders and the ones used in the common packages or tools, it is convenient to the users to use the functions from Microsoft Recommenders directly to extend or research the recommender algorithms.
- Compared to some open sourced software that may not have regular maintenance, Microsoft Recommenders is a community-driven repository with maintainers and contributors that are at the forefront of recommendation system development and implementation, including data scientists and researchers from Microsoft among others. All the code commits to the code base need to go through unit/integration tests to make sure that they all function well. This also requires that the developers who contribute codes to the repository write testing modules to conduct the tests on the codes. The benefit of doing such is that the codes in Microsoft Recommenders which are collected from the community are qualified.

The following subsections will introduce the development of an end-to-end recommender system with the open source code examples that are available in Microsoft Recommenders. Before doing the practices, it is required that the Microsoft Recommenders needs to be installed. Microsoft Recommenders can be run in multiple environment such as Windows, Linux, and MacOS. It supports Python 3.6 ˜Python 3.9.

If different Python versions are used, some packages may incur installation errors. It is a common practice to create a virtual environment by using conda or venv to isolate the packages to be installed. Here, it is assumed that the readers optional choose to do this on his/her own demand.

After finishing the prerequisites, Microsoft Recommenders can be configured by following the steps below.

Step 1, given that some of algorithm implementations in Microsoft Recommenders require code compiling, it is therefore needed to guarantee that the tools that are required for the compilation are pre-installed. For example, assuming the environment to install Microsoft Recommenders is Linux, build-essential should be installed to make sure that all the compiling tools can be properly used. This can be done by running the command as below in the Linux environment. (It is worth noting that if Windows system is used, this step can be replaced by installing Microsoft C++ Build Tools.)

```
sudo apt-get install -y build-essential
```

Step 2, after installing the compiling tools, the Python library of Microsoft Recommenders can be installed by using PyPI. The name of the corresponding Python library of Microsoft Recommenders is recommenders. Users can install and configure the library based on the actual needs about how to use it, by using the parameters in the pip installation. Recommenders support the installation parameters of examples, GPU, spark, dev, all, and experimental, and different parameters correspond to different installation configurations. The default is the minimum installation, i.e., examples. If the users want to use the recommender library with the setup of examples, the following pip install command can be used.

```
pip install --upgrade pip
pip install recommenders[examples]
```

Step 3, since Microsoft Recommenders supports various development environment, based on needs the users can choose to install libraries that are required by the running environment. It is worth mentioning that the Python libraries that are run in different environment have some special requirements that need to take care of.

- For the GPU versioned Microsoft Recommenders, since some relevant libraries (e.g., TensorFlow) require the acceleration toolkit by NVIDIA, users who use Microsoft Recommenders for GPU-based computation need to pre-install the CUDA-related toolkit. This can be done by the following.

  ```
  conda install cudatoolkit=10.0 "cudnn>=7.6"
  ```

- Microsoft Recommenders supports the Spark-based recommender system development. This can be done by configuring the parameters for Spark. Meanwhile, Microsoft Recommenders, the Spark version, is configurable and runnable on not merely the Spark cluster but also the standalone single node. For the Microsoft Azure users, the two use cases can be conducted conveniently on Azure Databricks or Azure Data Science Virtual Machine, respectively.

After the installation of Microsoft Recommenders, the Python library can be loaded inside the Python console by using the following command.

```
import recommenders
```

In the following subsections, the best practices in Microsoft Recommenders are presented with code examples.

6.3.1 Data Management and Preprocessing

Data management and preprocessing plays a vital role in constructing a recommender system. Due to the variety of recommender systems, at many times, the data preparation and preprocessing should closely follow the use cases that the recommenders system serves to. In addition, data transformation is also important in building recommender system. In this subsection, the data split and data transformation, which are the two most sub-tasks in data preparation in recommender system, will be introduced.

Data Split

Majorities of recommender systems are based on machine learning algorithms. Therefore, when building a recommender model, the raw data needs to be partitioned into the training set, validating set, and testing set. Compared to the conventional machine learning algorithms, the recommender system algorithms require special treatment due to its use case specific requirement. Hereafter, it is assumed the user behavior data in the recommender system is the most commonly one, which consists of four fundamental pieces of information, i.e., “user id,” “item id,” “rating (optional),” and “timestamp.” In the following examples, these four columns are selected from the raw data of MovieLens dataset. The utility function in Microsoft Recommenders can help download the MovieLens dataset and load them into the Pandas dataframe. Code examples for doing such are shown as below.

```
from recommenders.dataset.download_utils import \ maybe_download
import pandas as pd

# Here the MovieLens 100k dataset is used as an example.
```

```
DATA_URL = http://files.grouplens.org/datasets/movielens/ml-100k/u.data
DATA_PATH = "ml-100k.data"

COL_USER = "UserId"
COL_ITEM = "MovieId"
COL_RATING = "Rating"
COL_TIMESTAMP = "Timestamp"

filepath = maybe_download(DATA_URL, DATA_PATH)
data = pd.read_csv(filepath, sep="\t", names=[COL_USER, COL_ITEM, COL_RATING, COL_TIMESTAMP])

data_head()
```

Data format as read into the dataframe is shown below.

```
UserId MovieId Rating Timestamp
196            242            3              881250949
186            302            3              891717742
22            377            1             878887116
244            51            2             880606923
166            346            1              886397596
```

There are multiple ways of performing data split. Random split is simplest way of splitting data. This method of splitting randomly sample data points from the raw user behavior data by using the given split ratio.

```
from recommenders.dataset.python_splitters import python_random_split

data_train, data_test = python_random_split(data, ratio=0.7)

data_train.shape[0], data_test.shape[0]

# The split result has a ratio of 70%
(70000, 30000)
```

The split function python_random_split in Microsoft Recommenders also supports multiple ratios as input parameters. This makes it convenient to get multiple subsets, such as training set, validating set, and testing set, from the raw data. Also, when specifying the splitting ratio, the user can input any number even though they do not sum up to 1 – the function normalizes the ratio automatically according to the input number and perform the corresponding split. Example codes are shown below.

```
data_train, data_valid, data_test = python_random_split(
    data,
    ratio=[0.6, 0.2, 0.2]
)

data_train.shape[0], data_validate.shape[0], data_test.shape[0]

# The result split ratio is 60\%, 20\%, and 20\%.
(60000, 20000, 20000)
```

In many cases, data split in the recommender model building is performed based on user or item. The advantage of doing so is that it guarantees the users or items can consistently exist in both the training and the testing sets, which favors the model evaluating for some recommender algorithms. For example, for most of the collaborative filtering algorithms, it is very difficult if not impossible to perform recommendation for the “cold-start” users or items. As a result, when evaluating the model performance, if there are different user groups used in the evaluation, the comparison becomes unfair because the users-under-test may not even be used for building the model. The following code examples demonstrate how the function python_stratified_split in Microsoft Recommenders is used to perform data split by user. In the function call, in addition to the parameter of split ratio, the parameter of filter_by is also needed to specify whether the data is split by user or item. Also, to make sure that

there are at least certain number of users or items when performing the user-based or item-based splitting, there is another parameter min_rating for specifying the minimum interaction counts.

```
data_train, data_test = python_stratified_split(
    data,
    filter_by="user",
    min_rating=10,
    ratio=0.7,
    col_user=COL_USER,
    col_item=COL_ITEM
)
```

Sometimes it is common to consider the user-item interaction time when splitting the data based on either user or item. The objective of doing this is to make sure that when the training and testing a recommender model, the user behaviors in the testing set should all happen after those in the training set. By using the python_chrono_split function in Microsoft Recommenders, the data can be split on the scale of time, and the split is based on either user or item.

```
data_train, data_test = python_chrono_split(
    data,
    ratio=0.7,
    filter_by="user",
    col_user=COL_USER,
    col_item=COL_ITEM,
    col_timestamp=COL_TIMESTAMP
)
```

Checking the last ten rows and the first ten rows of the user's training set and testing set, respectively, it is shown below that the sequence of the user-item interactions is maintained correctly.

`data_train[data_train[COL_USER==1]].tail(10)`, we can get:

```
UserId MovieId Rating Timestamp
1              90            4             1997-11-03 07:31:40
1              219            1             1997-11-03 07:32:07
1              167            2             1997-11-03 07:33:03
1              162            4             1997-11-03 07:33:40
1              35            1             1997-11-03 07:33:40
1              230            4             1997-11-03 07:33:40

1              61            4             1997-11-03 07:33:40
1              265            4             1997-11-03 07:34:01
1              112            1             1997-11-03 07:34:01
1              57            5             1997-11-03 07:34:19
```

Printing `data_test[data_test[COL_USER==1]].head(10)`, we can get:

```
UserId MovieId Rating Timestamp
1              49            3             1997-11-03 07:34:38
1              30            3             1997-11-03 07:35:15
1              131            1             1997-11-03 07:35:52
1              233            2             1997-11-03 07:35:52
1              152            5             1997-11-03 07:36:29
1              82            5             1997-11-03 07:36:29
1              141            3             1997-11-03 07:36:48
1              72            4             1997-11-03 07:37:58
1              158            3             1997-11-03 07:38:19
1              33            4             1997-11-03 07:38:19
```

Data Transformation

In addition to data split, another important operation in the data preprocessing is data transformation. In the build-up of a recommender system, the data can be transformed in various ways. Many times the algorithm engineers and the data engineers should collaborate very closely to make sure the data is transformed properly in the entire system pipeline. In the following, the data transformation methods that are commonly used in recommender system are reviewed with code examples.

Majorities of the recommender systems are based on the implicit feedback of users. In the use cases of e-commerce, the examples of such implicit feedback include user clicks on items, user purchases, etc. In the use cases of video streaming or news feed, implicit feedback may refer to the lingering time that users stay on a media item. However, due to the fact that the implicit feedback usually does not provide the label for building a recommender model in a supervised approach, data transformation is therefore required to produce the necessary data labels from the implicit feedback information, such that the algorithmic model can be effectively trained. The commonly used methods for such operations are summarized as below.

This method mainly aggregate the users' implicit interactions with the items by counting the occurrences of user-item interaction. By using the MovieLens data, an aggregated dataset can be produced from one that does not have the explicit rating information. Codes for such operation are shown below.

```
data_count = data.groupby(['UserId', 'ItemId']).agg({'Timestamp':'count'}).reset_index()

# Here, the column of Affinity is to indicate the preferences of users.
data_count.columns = ['UserId', 'ItemId', 'Affinity']

UserId ItemId Affinity
1              1              2
1              2              3
2              1              2
2              2              2
2              3              1
3              1              1
3              3              4
```

In many circumstances, the count of the user-item interactions cannot precisely reflect the preferences of users. For example, on an e-commerce platform, some users may click an item for multiple times but eventually do not purchase it. The tendency to purchase of such users is therefore less than those who do not click many but purchase the item. In such case, the interactions between users and items can be weighted to indicate the importance of various interaction types. Codes of such weighted count of interactions are shown below.

```
data_w = data.copy()

conditions = [
    data_w['Type'] == 'click',
    data_w['Type'] == 'add',
    data_w['Type'] == 'purchase'
]

choices = [1, 2, 3]

data_w['Weight'] = np.select(conditions, choices, default = 'black')

# The data type of the weight information is changed to numeric.
data_w['Weight'] = pd.to_numeric(data_w['Weight'])

data_wcount = data_w.groupby(['UserId', 'ItemId'])['Weight'].sum().reset_index()
data_wcount.columns = ['UserId', 'ItemId', 'Affinity']

# The final data results are obtained as below
data_wcount
```

```
UserId    ItemId Affinity
1     1       2
1     2       5
2     1       3
2     2       6
2     3       3

3     1       1
3     3       7
```

The time-dependent count of interactions considers the time-scale impact of the interactions on top of the weighted importance of different types of interactions. One of the main rationales for such consideration is that users' behaviors drift over time. Therefore, when analyzing the preferences of users toward items, a common assumption is that a later interaction yields a more reliable indication. Such time-dependent counting method can be implemented by using the "decay" function. For example, in the following codes, an exponentially decayed function is applied to calculate the time-dependent count of interactions to indicate the user preferences.

```
# When using the time decay function, a base time point should be used as
reference.
T = 5
t_ref = pd.to_datetime(data_w['Timestamp']).max()

# The time decay is calculated in the numpy array. Here, the function to
# calculate time decay is based on the logarithm function.
data_w['Timedecay'] = data_w.apply(
    lambda x: x['Weight'] * np.power(0.5,
    (t_ref - pd.to_datetime(x['Timestamp'])).days / T), axis=1
)
# The user preferences can be obtained by summing up the time decay.
data_wt = data_w.groupby(['UserId', 'ItemId'])['Timedecay'].sum().reset_index()
data_wt.columns = ['UserId', 'ItemId', 'Affinity']

# The results are shown below data_w
UserId    ItemId  Affinity
1     1       1.319508
1     2       3.789291
2     1       2.400855
2     2       4.590914
2     3       2.611652
3     1       1.000000
3     3       5.883057
```

Negative sampling is a commonly used technique to sample the unseen data points from a given sampling space based on an appropriate assumption on its distribution characteristics. In the examples above, the method to generate the explicit feedback is based on the statistical properties of the user-item interactions. However, in many situation the method does not work properly due to the over simplification of the assumption. The theory of the negative sampling is that, if there are merely the positive feedback in the interaction data, by assuming that there must be some negative feedback in the items that the users have never interacted with before, the unseen items can be sampled with a predefined distribution and then added into the set of all the sample data. To use the example above, if only the positive feedback from the data are selected to form a dataset, it can be done by the following.

```
data_b = data[['UserId', 'ItemId']].copy()
data_b['Feedback'] = 1

# Here the duplicated user-item interactions are removed.
data_b = data_b.drop_duplicates()
data_b
UserId    ItemId  Feedback
```

```
1     1       1
1     2       1
2     1       1
2     2       1
2     3       1
3     3       1
3     1       1
```

The items that the users have not interacted with can then be obtained. Codes are shown as below.

```
# All the users and items are obtained in the first place.
users = data2['UserId'].unique() items = data2['ItemId'].unique()

# Get a user-item interaction cartesian product.
interaction_lst = []
for user in users:
    for item in items:
        interaction_lst.append([user, item, 0])

data_all = pd.DataFrame(
    data=interaction_lst,
    columns= ["UserId", "ItemId", "FeedbackAll"]
)

# Lastly, the positive and negative samples are integrated to generate a
# dataset that have both types of feedback.
data_ns = pd.merge(
    data_all,
    data2_b,
    on=['UserId', 'ItemId'],
    how= 'outer'
).fillna(0).drop('FeedbackAll', axis=1)

data_ns
UserId  ItemId  Feedback
1     1       1.0
1     2       1.0
1     3       0.0
2     1       1.0
2     2       1.0
2     3       1.0
3     1       1.0
3     2       0.0
3     3       1.0
```

Negative sampling can be done by using the function negative_feedback_sampler from the module recommenders.dataset.pandas_df_utils of Microsoft Recommenders. The function not only provides the interface to conveniently construct the negative sampled dataset but also allows configuration on the sampling ratio for the negative feedback.

6.3.2 Algorithm Selection and Model Training

Recommender algorithm and model are the key to building the recommender system. Microsoft Recommenders has collected more than 20 algorithms thus far. These algorithms include the classic ones like SVD, ALS, etc. as well as the recently advanced ones like xDeepFM, LightGCN, etc. Usually the algorithms should be selected based on the actual use case applications. It is common that the selection of recommender algorithms can be based on several criteria. Based on the purpose of recommendation, the algorithms can be categorized as collaborative filtering, content-based filtering, time-based model, etc. Based on the stage in a recommendation pipeline, the algorithms can be categorized into recall algorithm, re-ranking algorithm, personalized algorithm, etc. Based on

the implementation, algorithms can be categorized into the groups like CPU-based ones, GPU-based ones, deep learning-based ones, Spark-based ones, etc. The categorization helps the recommender system designer and developer to consider for selection at the early stage of the recommender system construction. For example, for an e-commerce platform, the recommender system it is to build should consider the following aspects.

First, the precision of the recommendation results. The precision is generally reflected by the relevancy of the recommendation results to the users' preferences. If the relevancy is high, the purchase rate of users will be high. This requires the selected algorithm performs well in generalizing the patterns it learns from the useful information in the historical data with which the recommendations can be generated. The algorithms for such purpose include SVD, FM/FFM, Wide & Deep, etc.

Second, the diversity of the recommendation results. In general, the items that are recommended have a larger size than the users. As a result, from the recommendations the users want to not merely get the relevant items of their interest but also have large enough exploration space to find relevant items, the latter of which is guaranteed by the diversity of the recommendation results. The recommendation algorithm can then be selected to aim at improving diversity. In the meantime, for the algorithm explainability, if the information provided by the recommender system can help the users to extend the search path or exploration space of new items, it may help provide ideal recommendation results. In such algorithms, the relevancy between users and items is analyzed to generate recommendations, and the graph technology is applied effectively to address the issue. Example algorithms in such group are DKN, RL-based recommendation algorithm, etc.

Third, the scalability of the recommender system. Due to the vast size of the user and item pool as well as the dynamics of the user-item interactions at the front-end of the recommender system, the recommender system for e-commerce needs to be scalable when it is deployed. This requirement restrains the application of many great algorithms that are published in the academic papers due to its limitation in scalable productionization. In addition, the system and infrastructure requirement for algorithm implementation should be considered carefully. The algorithms that are scalable are usually deployable within a framework where the scalable computing is availed. Examples of such algorithms include the ALS implementation of the SVD algorithm in Spark, Spark-based LightGBM, SAR+ from Microsoft, etc.

In real-world applications, algorithm selection is much more complex than the above overview. In many circumstances, engineering optimization is very important. Among all the engineering challenges, the one that tries to optimize the global objective understand various specifications is the biggest. For example, it is known that a complicated model is more prone to generalizing the recommendation performance, but some deep learning algorithms cannot be trained efficiently within the limited time frame (e.g., the time series based deep learning model). In such case, the engineers should adjust according to the actual needs from the business requirement. One of the techniques is to deploy the recommendation models hierarchically and apply different algorithms based on the engineering specifications and the algorithmic characteristics – at the recall stage, a simple yet effective algorithm is used to maximize the precision without much loss on accuracy; at the re-ranking or the personalized stage, a sophisticated algorithm is applied due to its desirable performance on the small candidate set. In summary, the algorithm selection process is a collaborative practice that requires the algorithm engineers, data scientists, business decision makers, etc. to take part in. Though the algorithm engineers play the vital role in the technological implementation, to make sure the global objective is optimized the holistic system as well as its impact on business should be considered at large.

After the selection of algorithms, the key task next is to optimize the model performance in the training process. For the algorithms implemented in different platforms, the process for model training is more or less the same. The algorithm engineers and the architects need to set the metrics for evaluating the model and tune the parameters to achieve the optimal training outcome. Nowadays, there are many tools that make the process automatic. For the machine learning based algorithms, especially the deep learning based ones, parameter tuning becomes a time-consuming yet very important task for model building. In Microsoft Recommenders, the code examples demonstrate how to use the common tool such as NNI, Azure Machine Learning Service, hyperopt, etc., for effective parameter tuning. Due to the complexity the code examples are not detailed here. They can be referenced under the directory of examples in Microsoft Recommenders, in the sub-directory of 04_model_select_and_optimize, where the tools for tuning algorithms like SVD, ALS, NCF, etc. can be found.

In the following part, three different types of algorithms with practices are introduced for building a recommender model.

Spark-Based ALS

The collaborative filtering recommendation algorithm that uses the ALS method is widely used in many application scenarios due to its advantage in maintenance and deployment. In the real-world industry applications,

considering the need for scalability, the ALS-based recommendation model is able to handle large-scale user-item interaction data, to effectively generate the recommendation results. The Spark-based implementation of the ALS method further leverages the high-scalable and high-performance characteristics of the Spark framework such that in many collaborative filtering use cases it can deal with the large-scale recommendation problems [2, 4, 8].

Using the Spark-based ALS implementation is similar to other matrix factorization method. The algorithm is applicable to the data where the user-item interactions are available. The interactions can be either implicit or explicit. Considering the distributed computing framework of Spark, the method can be applied for the use case (e.g., recall stage of majorities of the recommender systems) where dataset is large scale and the requirement of the computational efficiency is high. In addition, given that the Spark framework is commonly used for data preprocessing in the entire data pipeline of a recommender system, the recommendation models that are based on Spark can be efficiently used to interface with the processed data in the same Spark platform, such that the data transformation can be saved to guarantee the consistency of data flow.

To use the Spark-based ALS, the Apache Spark framework should be installed as a prerequisite. The programming language used in the same chapter is Python, so to use Spark PySpark with the right version should be installed. The information for installation can be found in the website of Microsoft Recommenders. In the following code examples, the Spark-based ALS method is demonstrated for analyzing and modeling with the MovieLens dataset.

First, the modules in PySpark and Microsoft Recommenders that are used in the example are imported.

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
import pyspark.sql.functions as F
from pyspark.sql.functions import col
from pyspark.ml.tuning import CrossValidator
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import FloatType, IntegerType, LongType

from recommenders.datasets import movielens
from recommenders.utils.spark_utils import start_or_get_spark
from recommenders.evaluation.spark_evaluation import SparkRankingEvaluation,
SparkRatingEvaluation
from recommenders.tuning.parameter_sweep import generate_param_grid
from recommenders.datasets.spark_splitters import spark_random_split
```

Second, the global variables used for assisting model building are defined.

```
# Define the features used in the data.
COL_USER = "UserId"
COL_ITEM = "MovieId"
COL_RATING = "Rating"
COL_PREDICTION = "prediction"
COL_TIMESTAMP = "Timestamp"

# Define the data type information used in the Spark data.
schema = StructType(
    (
        StructField(COL_USER, IntegerType()),
        StructField(COL_ITEM, IntegerType()),
        StructField(COL_RATING, FloatType()),
        StructField(COL_TIMESTAMP, LongType()),
    )
)

# Define the Spark ALS model parameters
RANK = 10 MAX_ITER = 15 REG_PARAM = 0.05
```

```
# The number of items to recommend
K = 10
```

Next, a new Spark session is created. The MovieLens data is imported. Here, the data loading function in Microsoft Recommenders is used.

```
# Use the start_or_get_spark function in Microsoft Recommenders to create a new
Spark session.
spark = start_or_get_spark("ALS Deep Dive", memory="16g")

# Using the existing Spark session object to read the MovieLens data into the
Spark Dataframe.
dfs = movielens.load_spark_df(spark=spark, size="100k", schema=schema)

dfs.show(5)
+------+-------+------+---------+
|UserId|MovieId|Rating|Timestamp|
+------+-------+------+---------+
|   196|    242|   3.0|881250949|
|   186|    302|   3.0|891717742|
|    22|    377|   1.0|878887116|
|   244|     51|   2.0|880606923|
|   166|    346|   1.0|886397596|
+------+-------+------+---------+
only showing top 5 rows
```

The data imported into the Spark session is split randomly into the training set and the testing set. Here the split ratio for the training and test sets is 75% to 25%.

```
    dfs_train, dfs_test = spark_random_split(dfs, ratio=0.75, seed=42)
```

The data after splitting is used for building a Spark ALS model. It is worth mentioning that the data is represented as Spark DataFrame in the model building process. Therefore, correspondingly, the ALS module in pyspark.ml is used for model training. Particularly, the parameter of coldStartStrategy in the ALS class is set to “drop,” such that the even if the users or items are not purposely kept in both the training set and testing set at the same, the evaluation of the trained ALS model can still be fairly conducted. This is because, when the Spark ALS coldStartStrategy parameter is set to “drop,” the “cold-start” values of either users or items are neglected so that the training set and the testing set contain the same group of users or items, which guarantees the fairness of the model evaluation.

```
als = ALS(
    maxIter=MAX_ITER,
    rank=RANK,
    regParam=REG_PARAM,
    userCol=COL_USER,
    itemCol=COL_ITEM,
    ratingCol=COL_RATING,
    coldStartStrategy="drop"
)

model = als.fit(dfs_train)
```

In the end, the model that has been trained is evaluated on the testing data. The following codes demonstrate how the trained ALS model generates scores and then compare the scores against the ground truth in the testing set to produce the evaluation results.

```
dfs_pred = model.transform(dfs_test).drop(COL_RATING)
```

```
evaluations = SparkRatingEvaluation(
    dfs_test,
    dfs_pred,
    col_user=COL_USER,
    col_item=COL_ITEM,
    col_rating=COL_RATING,
    col_prediction=COL_PREDICTION
)

print( "RMSE score = {}".format(evaluations.rmse()), "MAE score =
    {}".format(evaluations.mae()), "R2 score =
    {}".format(evaluations.rsquared()), "Explained variance score =
    {}".format(evaluations.exp_var()), sep="\n"
)
RMSE score = 0.9697095550242029
MAE score = 0.7554838330206419
R2 score = 0.24874053010909036
Explained variance score = 0.2547961843833687
```

The Implementation of the Sequential Recommender Model

In Chap. 4.4, the sequential recommender is introduced. The user behavior data is usually ordered on the time basis. The analysis on the user sequences is not merely useful to exploit the pattern of how the users' interest evolve but also is beneficial in capturing the short-term and long-term interests of users. For example, if a user frequently visits the sites to check the mobile phone items, it indicates that the user has interest in purchasing a mobile phone recently. For the short-term interesting, the more recent the behavior is, the more information it can reflect. In comparison, the long-term interest of the users reflects the general preferences of the users, and it is rather stationary over time. Sequential recommendation is a key branch of the recommender system technology. It has been widely studied in academia and applied in the real-world industrial applications. For example, Shumpei Okura proposed the GPU-based model to analyze the user histories on the Yahoo Japan news recommendation platform [10]. Guorui Zhou proposed an improved version of the GRU model and used it in the Alibaba advertisement recommendation [11, 12]. GRU is a simple yet efficient method for sequential recommendation, and it has already become the baseline for the following sequential recommendation algorithms. In this section, the GRU algorithm for building sequential recommender model is demonstrated. The completed code examples can be found in the code base of Microsoft Recommenders, in the Jupyter notebook of examples/00_quick_start/sequantil_recsys_amazondataset.ipynb.

First, import the libraries used in the example.

```
import sys
import os
import logging
import scrapbook as sb
import tempfile
from recommenders.utils.timer import Timer
from recommenders.utils.constants import SEED
from recommenders.models.deeprec.deeprec_utils import prepare_hparams
from recommenders.datasets.amazon_reviews import download_and_extract,
data_preprocessing
from recommenders.datasets.download_utils import maybe_download

from recommenders.models.deeprec.models.sequential.gru4rec import GRU4RecModel from recommenders.models
import SequentialIterator
```

Create a new temporary directory to preserve the config file, the input data and the output files.

```
tmp_path = tempfile.mkdtemp()
data_path = os.path.join(tmp_path, 'gru4rec')
```

The data_path variable can be printed to check the detailed path information. The path is generated automatically by the system, e.g., /tmp/tmpxjq4i3ij/gru4rec

For the sake of conveniently managing the parameters, the parameters and variables that are used for model definition and model training are saved into the config file. During the model training process, the config file is loaded. Here, it is necessary to create a file named gru4rec.yaml under data_path.

```
yaml_file = os.path.join(data_path, 'gru4rec.yaml')
```

Write the following content into the yaml file.

```
data:
    user_vocab : ./tests/resources/deeprec/gru4rec/user_vocab.pkl # the map file of user to id
    item_vocab : ./tests/resources/deeprec/gru4rec/item_vocab.pkl # the map file
of item to id
    cate_vocab : ./tests/resources/deeprec/gru4rec/category_vocab.pkl # the map file of
category to id

model:
    method : classification # classification or regression
    model_type : GRU4Rec
    layer_sizes : [100, 64]  # layers' size of DNN. In this example,
    DNN has two layers, and each layer has 100 hidden nodes.
    activation : [relu, relu] # activation function for DNN
    user_dropout: True
    dropout : [0.3, 0.3]  #drop out values for DNN layer
    item_embedding_dim : 32 # the embedding dimension of items
    cate_embedding_dim : 8  # the embedding dimension of categories
    user_embedding_dim : 16 # the embedding dimension of users

train:
    init_method: tnormal  # method for initializing model parameters
    init_value : 0.01 # stddev values for initializing model parameters
    embed_l2 : 0.0001 # l2 regularization for embedding parameters
    embed_l1 : 0.0000 # l1 regularization for embedding parameters
    layer_l2 : 0.0001 # l2 regularization for hidden layer parameters
    layer_l1 : 0.0000 # l1 regularization for hidden layer parameters
    cross_l2 : 0.0000  # l2 regularization for cross layer parameters
    cross_l1 : 0.000   # l1 regularization for cross layer parameters
    learning_rate : 0.001
    loss : softmax  # pointwise: log_loss, cross_entropy_loss, square_loss
    pairwise: softmax
    optimizer : lazyadam  # adam, adadelta, sgd, ftrl, gd, padagrad, pgd, rmsprop, lazyadam
    epochs : 50  # number of epoch for training

   batch_size : 400  # batch size, should be constrained as an integer multiple of the number of (1 +
    True
    enable_BN : True # whether to use batch normalization in hidden layers
    EARLY_STOP : 10 # the number of epoch that controls EARLY STOPPING
    max_seq_length : 50 # the maximum number of records in the history sequence
    hidden_size : 40 # the shape of hidden size used in RNN
    need_sample: True # whether to perform dynamic negative sampling in mini-batch
    train_num_ngs: 4 # indicates how many negative instances followed by one positive instances if nee

info:
    show_step : 100  # print training information after a certain number of mini-batch
    save_model: True  # whether to save models
    save_epoch : 1  # if save_model is set to True, save the model every save_epoch.
```

```
  metrics : ['auc','logloss']  # metrics for evaluation.

 pairwise_metrics : ['mean_mrr', 'ndcg@2;4;6', "group_auc"]  # pairwise metrics for evaluation, avai
  needed
  MODEL_DIR : ./tests/resources/deeprec/gru4rec/model/gru4rec_model/  # directory of saved models.
  SUMMARIES_DIR : ./tests/resources/deeprec/gru4rec/summary/gru4rec_summary/  # directory of saved s
  write_tfevents : True  # whether to save summaries.
```

It can be observed that the variables in the yaml file are about the data path, model parameters, training process, and data output. The readers can adjust the parameters based on actual need.

The dataset used in the experiment is the Movies and TV subset in the Amazon Dataset, which is widely used in the research and study of recommender system.

Microsoft Recommenders provides a few built-in methods for processing the common datasets. These methods are collected under the directory of recommenders/datasets, where the preprocessing functions for the Amazon dataset are available. The function `download_and_extract` can automatically download the Amazon dataset from the data source website and unpack the raw data files. As usual the raw data is split into the training set, the validating set, and the testing set. The training set is used for training the model, the validating set is to verify whether the model parameters are converged, and the testing set is to test the model performance and it cannot be used for the training process but only for evaluating the trained model. In the raw Amazon dataset, the samples are all positive, so certain amount of the negative samples should be generated. In the example here, the ratio between the positive and the negative samples in both the training set and the validating set is 1 to 4 (that is, the variables of `train_num_ngs` and `valid_num_ngs`), whilst that in the testing set is 1 to 9 (i.e., the variable of `test_num_ngs`). To quickly check whether the program can be run properly, a small dataset is created. In the codes below, the variable of sample_rate is to control the sampling proportion of the data used for conducting the experiment from the raw data. When the entire raw data is used, the parameter can be set to be 1.

```
train_file = os.path.join(data_path, r'train_data')
valid_file = os.path.join(data_path, r'valid_data')
test_file = os.path.join(data_path, r'test_data')
user_vocab = os.path.join(data_path, r'user_vocab.pkl')
item_vocab = os.path.join(data_path, r'item_vocab.pkl')
cate_vocab = os.path.join(data_path, r'category_vocab.pkl')
output_file = os.path.join(data_path, r'output.txt')
train_num_ngs = 4
valid_num_ngs = 4
test_num_ngs = 9
sample_rate = 0.01
input_files = [reviews_file, meta_file, train_file, valid_file, test_file,
user_vocab, item_vocab, cate_vocab]
data_preprocessing(*input_files, sample_rate=sample_rate,
valid_num_ngs=valid_num_ngs, test_num_ngs=test_num_ngs)
```

Next, a variable can be created to preserve all the hyperparameters used for building the model. These parameters are previously stored in the `yaml_file`. If setting the parameters explicitly when conducting the experiment by calling the `prepare_hparams` function, the values of these parameters will overwrite the ones in `yaml_file`.

```
hparams = prepare_hparams(
  yaml_file,
  embed_l2=0.,
  layer_l2=0.,
  learning_rate=0.001,
  epochs=EPOCHS, batch_size=BATCH_SIZE, show_step=20,
  MODEL_DIR=os.path.join(data_path, "model/"),
  SUMMARIES_DIR=os.path.join(data_path, "summary/"), user_vocab=user_vocab,
  item_vocab=item_vocab, cate_vocab=cate_vocab, need_sample=True,
  train_num_ngs=train_num_ngs,
```

```
)
```

Here, a data loader is declared.

```
input_creator = SequentialIterator
```

Usually the data used in the industry-grade recommender system has quite large scale. Therefore, it is hard to load the data into the memory all at once. The object of `SequentialIterator` is essentially an iterator, and it loads data from the files in batches that have sizes of `batch_size` into memory, and it then converts it into the matrix or tensor which is readable by TensorFlow. The loaded data is then returned to the model object. The optimization space for the data loader is quite large. For example, to further improve the efficiency of the program, the producer-consumer pattern can be leveraged. That is, the data loader becomes a producer, and it produced batches of data continuously for the model to run on the training process. This process is usually performed on the CPU devices. On the other side, the model object can be treated as a consumer. It takes the data for deriving the forward propagation in the neural network and then updating the parameters against the gradient descent. This process is on the GPU devices. If the producer and the consumer can be parallelized in the pipeline, the waiting time for the entire process can be greatly reduced. In the tutorial of this section, for simplicity purpose, the operations for such optimization are not discussed. In the Amazon dataset, the unique IDs of items and its class ID are concatenated to represent a single item. Each user uses the maximumly `max_seq_length` times of behavior for building the model, and for those users who do not have such amount of behavior the vector will be padded with empty value.

After the preparation work, based on the predefined parameters and data loader, a sequential model based on the GRU algorithm can be created.

```
model = GRU4RecModel(hparams, input_creator, seed=RANDOM_SEED)
```

The theory of the GRU4Rec algorithm is quite simple. It concatenates the vectors of the user historical behavior on the item ID as well as its group ID, to feed into a GRU layer to obtain the intrinsic vectors of the users.

```
with tf.name_scope("gru"):
    self.mask = self.iterator.mask
    self.sequence_length = tf.reduce_sum(self.mask, 1)
    self.history_embedding = tf.concat(
        [self.item_history_embedding, self.cate_history_embedding], 2)
    rnn_outputs, final_state = dynamic_rnn(GRUCell(self.hidden_size),
        inputs=self.history_embedding, sequence_length=self.sequence_length,
        dtype=tf.float32, scope="gru",)
    tf.summary.histogram("GRU_outputs", rnn_outputs)
    return final_state
```

Given that the different users may have different length of historical data, the object of `dynamic_rnn` can compute only on the effective users and neglect those with the empty values due to the requirement of the data format consistency. In the beginning, the model parameters are at the initial state and they do not have the differentiation power. To verify this, the initialized model can be used on the testing dataset for scoring.

```
model.run_eval(test_file, num_ngs=test_num_ngs)
```

Output:

```
{'auc': 0.4857, 'logloss': 0.6931, 'mean_mrr': 0.2665, 'ndcg@2': 0.1357,
'ndcg@4': 0.2186, 'ndcg@6': 0.2905, 'group_auc': 0.4849}
```

It can be seen that the evaluation metrics of auc and group_auc are both close to 0.5, i.e., an indication of a random guess. If the model is iteratively trained for ten times on the training set,

```
model = model.fit(train_file, valid_file, valid_num_ngs=valid_num_ngs)
```

The output will be

```
eval valid at epoch 1:
auc:0.4975,logloss:0.6929,mean_mrr:0.4592,ndcg@2:0.3292,ndcg@4:0.5125,ndcg@6:0.5915,group_auc:0.4994
eval valid at epoch 2:
auc:0.6486,logloss:0.6946,mean_mrr:0.5567,ndcg@2:0.472,ndcg@4:0.6292,ndcg@6:0.6669,group_auc:0.6363
eval valid at epoch 3:
auc:0.6887,logloss:0.8454,mean_mrr:0.6032,ndcg@2:0.537,ndcg@4:0.6705,ndcg@6:0.7022,group_auc:0.683
eval valid at epoch 4:
auc:0.6978,logloss:0.7005,mean_mrr:0.6236,ndcg@2:0.5622,ndcg@4:0.6881,ndcg@6:0.7175,group_auc:0.699
eval valid at epoch 5:
auc:0.7152,logloss:0.6695,mean_mrr:0.6382,ndcg@2:0.582,ndcg@4:0.7009,ndcg@6:0.7286,group_auc:0.7139
eval valid at epoch 6:
auc:0.722,logloss:0.6141,mean_mrr:0.637,ndcg@2:0.5796,ndcg@4:0.6993,ndcg@6:0.7276,group_auc:0.7116
eval valid at epoch 7:
auc:0.7287,logloss:0.6183,mean_mrr:0.6417,ndcg@2:0.5875,ndcg@4:0.7031,ndcg@6:0.7312,group_auc:0.7167
eval valid at epoch 8:
auc:0.7342,logloss:0.6584,mean_mrr:0.6538,ndcg@2:0.6006,ndcg@4:0.7121,ndcg@6:0.7402,group_auc:0.7248
eval valid at epoch 9:
auc:0.7324,logloss:0.6268,mean_mrr:0.6541,ndcg@2:0.5981,ndcg@4:0.7129,ndcg@6:0.7404,group_auc:0.7239
eval valid at epoch 10:
auc:0.7369,logloss:0.6122,mean_mrr:0.6611,ndcg@2:0.6087,ndcg@4:0.7181,ndcg@6:0.7457,group_auc:0.731
```

It can be seen that along with the increase of the epoch, the convergency of the model on the training set becomes gradually better. After the model has been trained, it can be used to score on the testing set.

```
model.run_eval(test_file, num_ngs=test_num_ngs)
```

Output:

```
{'auc': 0.7174, 'logloss': 0.6149, 'mean_mrr': 0.4835, 'ndcg@2': 0.3939,
'ndcg@4': 0.4982, 'ndcg@6': 0.5503, 'group_auc': 0.7073}
```

To minimize the impact of the randomness in each experiment, multiple experiments are needed to calculate the statistical mean and variance, with which the hypothesis testing should be conducted to make sure the conclusion is sound.

Knowledge Graph-Based Recommender System

Chapter 4.5 of the book introduced the knowledge graph-based recommender system. A knowledge graph collects large volume of relational information that exists in the world. This information not merely enriches the descriptions of items but also creates the vast connections between item entities. With the assistance of the knowledge graph, it is easy and precise to build model for the user-item relationships. This subsection takes the use case of the academic paper recommendation as an example to practically illustrate the development of a DKN recommender model. The task of the academic paper recommendation is to predict the papers that authors will cite, based on the analysis on the historical citations that the authors have. The dataset used in the experiment is from the Microsoft Academic Graph. The whole codes used in the illustration can be found in the directory of recommenders/examples/07_tutorials/KDD2020-tutorial. The experiment can be partitioned into preparation, pre-training word and entity representation, and model training.

The codes and data used in the experiment need to be prepared in the first place. A new directory for the experiment, namely KRS, can be created and used as the working directory. The contents under recommenders/examples/07_tutorials/KDD2020-tutorial/utils can be copied into the directory of KRS, because the codes under utils keep the file processing codes that are used in the experiment. Next, the raw dataset used for such experiment is downloaded and unpacked.

```
wget https://recodatasets.z20.web.core.windows.net/kdd2020/data_folder.zip unzip
data_folder.zip -d data_folder
```

Import the dependencies used in the experiment.

```
import os
import pickle
import time
from utils.task_helper import *
from utils.general import *
from utils.data_helper import *
```

The core idea of the DKN algorithm is to leverage the knowledge entities to augment the representation of the academic papers. The following shows the format of paper representation.

```
[Newsid] [w1, w2, w3...wk] [e1, e2, e3...ek]
```

`Newsid` is the ID for the paper, w1, w2, w3, …, wk are the word ID in the paper titles, `e1, e2, e3, …, ek` are the corresponding IDs of the entities. If the corresponding words are not entities, they are padded with zero. To favor the one-hot encoding of the IDs which are used as model input, the word and entity IDs are encoded into the integers that start with 1.

```
InFile_dir = 'data_folder/raw'
OutFile_dir = 'data_folder/my'
create_dir(OutFile_dir)

Path_PaperTitleAbs_bySentence = os.path.join(InFile_dir,
'PaperTitleAbs_bySentence.txt')
Path_PaperFeature = os.path.join(OutFile_dir,
'paper_feature.txt')
max_word_size_per_paper = 15

word2idx = {}
entity2idx = {}
relation2idx = {}

word2idx, entity2idx = gen_paper_content(Path_PaperTitleAbs_bySentence, Path_PaperFeature, word2idx, en
field=["Title"], doc_len=max_word_size_per_paper )
```

In a knowledge graph, the entity is a tuple of head entity, tail entity, relationship. Therefore, the data files should be generated to maintain such tuple representation of the entities.

```
Path_RelatedFieldOfStudy = os.path.join(InFile_dir, 'RelatedFieldOfStudy.txt')
OutFile_dir_KG = os.path.join(OutFile_dir, 'KG')

create_dir(OutFile_dir_KG)
gen_knowledge_relations(Path_RelatedFieldOfStudy, OutFile_dir_KG, entity2idx,
relation2idx)
```

Usually, the Word2Vec algorithm is used to pre-train the words. The pre-training needs to load a collection of the sentences and make use of the mutual relationship between words in the sentences, to learn the meaningful word representations in vectors. The following codes are used to generate the corpus for the above purpose.

```
Path_SentenceCollection = os.path.join(OutFile_dir, 'sentence.txt')
gen_sentence_collection(Path_PaperTitleAbs_bySentence, Path_SentenceCollection,
word2idx )

word2idx_filename = os.path.join(OutFile_dir, 'word2idx.pkl')
entity2idx_filename = os.path.join(OutFile_dir, 'entity2idx.pkl')

with open(word2idx_filename, 'wb') as f:
```

```
    pickle.dump(word2idx, f)

dump_dict_as_txt(word2idx, os.path.join(OutFile_dir, 'word2id.tsv'))

with open(entity2idx_filename, 'wb') as f:
    pickle.dump(entity2idx, f)
```

Since the task is to predict the author citations based on the historical ones, the historical data needs to be arranged. This can be done in three main steps: load the authors and their publications in the past, load the relationships between the publications, and based on the first two steps infer the citations that the authors have.

```
Path_PaperReference = os.path.join(InFile_dir, 'PaperReferences.txt')
Path_PaperAuthorAffiliations = os.path.join(InFile_dir, 'PaperAuthorAffiliations.txt')
Path_Papers = os.path.join(InFile_dir, 'Papers.txt')
Path_Author2ReferencePapers = os.path.join(OutFile_dir, 'Author2ReferencePapers.tsv')
author2paper_list = load_author_paperlist(Path_PaperAuthorAffiliations)
paper2date = load_paper_date(Path_Papers)
paper2reference_list = load_paper_reference(Path_PaperReference)

author2reference_list = get_author_reference_list(author2paper_list, paper2reference_list, paper2date)
output_author2reference_list(author2reference_list, Path_Author2ReferencePapers)
```

In the last, the training set, the validating set, and the testing set need to be generated. Usually, the papers that the authors have cited are ordered in time. For the time sequence of each author, the last paper item is put into the testing set, the second last is put into the validating set, and the remaining ones are put into the training set.

```
OutFile_dir_DKN = os.path.join(OutFile_dir, 'DKN-training-folder')
create_dir(OutFile_dir_KG)

gen_experiment_splits( Path_Author2ReferencePapers, OutFile_dir_DKN, Path_PaperFeature, item_ratio=0.1,
```

To make the debugging convenient, a small dataset is generated by using the parameter of `item_ratio=0.1`, which indicates that 10% of the raw data is sampled. Also, the file names in the generated dataset all contain a tag of "small." If the whole dataset is needed, the parameter and tag value of `item_ratio=1` and "full" can be used.

A good practice before training the DKN model is to pre-train the fundamental components such as word and entity representation. This is beneficial to allow the model to thoroughly leverage the external general knowledge and make the training process converge quickly. For word pre-training, the Word2Vec module from the gensim library is used. Before the pre-training, the dependency library is loaded, and the iterator for the sentence corpus is defined.

```
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import Word2Vec
import time
from utils.general import *
import numpy as np
import pickle
from utils.task_helper import *

class MySentenceCollection:
    def __init__(self, filename):
        self.filename = filename
        self.rd = None

    def __iter__(self):
        self.rd = open(self.filename, 'r', encoding='utf-8', newline='')
        return self
```

```
    def __next__(self):
        line = self.rd.readline()
        if line:
            return list(line.strip(’’).split(’ ’))
        else:
            self.rd.close()
            raise StopIteration
```

Next, the algorithm in Word2Vec can be used to pre-train the words.

```
def train_word2vec(Path_sentences, OutFile_dir):
    OutFile_word2vec = os.path.join(OutFile_dir, r’word2vec.model’)
    OutFile_word2vec_txt = os.path.join(OutFile_dir, r’word2vec.txt’)
    create_dir(OutFile_dir)
    my_sentences = MySentenceCollection(Path_sentences) model =
    Word2Vec(my_sentences, size=32, window=5, min_count=1, workers=8, iter=10)
    model.save(OutFile_word2vec)
    model.wv.save_word2vec_format(OutFile_word2vec_txt, binary=False)

InFile_dir = ’data_folder/my’
OutFile_dir = ’data_folder/my/pretrained-embeddings’
Path_sentences = os.path.join(InFile_dir, ’sentence.txt’)

train_word2vec(Path_sentences, OutFile_dir)
```

For the knowledge graph pre-training, the simplest method of TransE is used [5]. The following shell command is executed to get the embeddings of the knowledge graph entities.

```
echo $PWD
cd data_folder
git clone https://github.com/thunlp/Fast-TransX.git
cd Fast-TransX
cd transE
g++ transE.cpp -o transE -pthread -O3 -march=native
inpath="../../my/KG/"
outpath="../../my/KG/"
if [ ! -d $outpath ]; then
    mkdir -p $outpath;
fi ./transE -size 32 -sizeR 32 -input $inpath  -output  $outpath
-epochs 10 -alpha 0.001
```

In the DKN model, not only the entity embeddings are used for model building, the context embedding that represent the auxiliary information about the entities are also used. To further proceed with the follow-up operations, the entity embeddings and the word embeddings are converted to the numpy arrays, which favors the pre-loading in the DKN model.

```
OutFile_dir_KG = ’data_folder/my/KG’
OutFile_dir_DKN = ’data_folder/my/DKN-training-folder’
EMBEDDING_LENGTH = 32
entity_file = os.path.join(OutFile_dir_KG, ’entity2vec.vec’)
context_file = os.path.join(OutFile_dir_KG, ’context2vec.vec’)
kg_file = os.path.join(OutFile_dir_KG, ’train2id.txt’)

gen_context_embedding(entity_file, context_file, kg_file, dim=EMBEDDING_LENGTH)

load_np_from_txt(
    os.path.join(OutFile_dir_KG, ’entity2vec.vec’),
    os.path.join(OutFile_dir_DKN, ’entity_embedding.npy’),
```

```
)

load_np_from_txt(
    os.path.join(OutFile_dir_KG, 'context2vec.vec'),
    os.path.join(OutFile_dir_DKN, 'context_embedding.npy'),
)

format_word_embeddings(
    os.path.join(OutFile_dir, 'word2vec.txt'),
    os.path.join(InFile_dir, 'word2idx.pkl'),

    os.path.join(OutFile_dir_DKN,
    'word_embedding.npy')
)
```

After the preparation work as mentioned above, a DKN model can be trained. The complete code implementation of DKN model can be found in recommenders/models/deeprec/models/dkn.py. The contents are too long to be presented here. Here only the core function Kim_CNN that fuses the knowledge text convolution network is shown below.

```
def _kims_cnn(self, word, entity, hparams):
```

The variables of `word` and `entity` represent the index of the word and entity. They can be used to get the embeddings from the Embedding Table.

```
if hparams.use_entity and hparams.use_context:
    entity_embedded_chars = tf.nn.embedding_lookup(self.entity_embedding, entity)
    context_embedded_chars = tf.nn.embedding_lookup(self.context_embedding, entity)
    concat = tf.concat(
        [embedded_chars, entity_embedded_chars, context_embedded_chars], axis=-1
    )
elif hparams.use_entity:
    entity_embedded_chars = tf.nn.embedding_lookup(
        self.entity_embedding, entity
    )
    concat = tf.concat(
        [
            embedded_chars, entity_embedded_chars
        ], axis=-1
    )
else:
    concat = embedded_chars
    concat_expanded = tf.expand_dims(concat, -1)
```

In the text, the representation of the words consists of word vector, entity vector, and context vector. The hyper parameter `hparams.user_entity` and `hparams.user_context` are used for ablation experimentation. They can be used to control whether to add the entity embedding vector or the context embedding vector. Following that, the word representation is added into the convolution operation to get the sentence-level representations.

```
pooled_outputs = []

for i, filter_size in enumerate(filter_sizes):
    with tf.compat.v1.variable_scope(
        "conv-maxpool-%s" % filter_size, initializer=self.initializer
    ):
        if hparams.use_entity and hparams.use_context:
            filter_shape = [filter_size, dim * 3, 1, num_filters]
```

```
        elif hparams.use_entity:
            filter_shape = [filter_size, dim * 2, 1, num_filters]
        else:
            filter_shape = [filter_size, dim, 1, num_filters]

        W = tf.compat.v1.get_variable(
            name="W" + "_filter_size_" + str(filter_size),
            shape=filter_shape,
            dtype=tf.float32,
            initializer=tf.contrib.layers.xavier_initializer(uniform=False),
        )
        b = tf.compat.v1.get_variable(
            name="b" + "_filter_size_" + str(filter_size),
            shape=[num_filters], dtype=tf.float32,
        )
        if W not in self.layer_params:
            self.layer_params.append(W)
        if b not in self.layer_params:
            self.layer_params.append(b)
        conv = tf.nn.conv2d(
            concat_expanded,
            W,
            strides=[1, 1, 1, 1],
            padding="VALID",
            name="conv",
        )
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        pooled = tf.nn.max_pool2d(
            h,
            ksize=[1, hparams.doc_size - filter_size + 1, 1, 1],
            strides=[1, 1, 1, 1],
            padding="VALID",
            name="pool",
        )
        pooled_outputs.append(pooled)
```

`filter_sizes` is the size of the convolution network kernels. It is recommended to use [1, 2, 3] in the DKN model, to represent that there are three different convolution kernels with the sizes of 1, 2, and 3. W and b are the weight parameter and the bias parameter of the kernel. For each kernel, there is a max pooling layer applied on the output vector. Given that there are totally `num_filter * num_filter_sizes` kernels, the size of the embedding vector of the sentences generated from `_kims_cnn` is `num_filter * num_filter_sizes`.

```
self.num_filters_total = num_filters * len(filter_sizes)
h_pool = tf.concat(pooled_outputs, axis=-1)
h_pool_flat = tf.reshape(h_pool, [-1, self.num_filters_total])
```

The process of experiment for DKN is similar to that in GRU. That is, the dependency module import, path definition, config file preparation, data iterator declaration, model training, etc. In the first place, the dependencies are loaded.

```
from recommenders.models.deeprec.deeprec_utils import *
from recommenders.models.deeprec.models.dkn import *
from recommenders.models.deeprec.io.dkn_iterator import *
import tensorflow as tf
```

Next, the paths of the data-dependent files are defined. These mainly include the pre-trained word vectors and the entity vectors as well as the training set, the validating set, and the testing set.

```
tag = 'small'
data_path = 'data_folder/my/DKN-training-folder'
yaml_file = './dkn.yaml'
train_file = os.path.join(data_path, r'train_{0}.txt'.format(tag))
valid_file = os.path.join(data_path, r'valid_{0}.txt'.format(tag))
test_file = os.path.join(data_path, r'test_{0}.txt'.format(tag))
user_history_file = os.path.join(data_path, r'user_history_{0}.txt'.format(tag))
news_feature_file = os.path.join(data_path, r'../paper_feature.txt')
wordEmb_file = os.path.join(data_path, r'word_embedding.npy')
entityEmb_file = os.path.join(data_path, r'entity_embedding.npy')
contextEmb_file = os.path.join(data_path, r'context_embedding.npy')
infer_embedding_file = os.path.join(data_path, r'infer_embedding.txt')
```

Then the data iterator and the model objects are declared.

```
hparams = prepare_hparams(
    yaml_file,
    news_feature_file=news_feature_file,
    user_history_file=user_history_file,
    wordEmb_file=wordEmb_file,
    entityEmb_file=entityEmb_file,
    contextEmb_file=contextEmb_file,
    epochs=5,
    is_clip_norm=True,
    max_grad_norm=0.5,
    history_size=20,
    MODEL_DIR=os.path.join(data_path, 'save_models'),
    learning_rate=0.001,
    embed_l2=0.0,
    layer_l2=0.0,
    use_entity=True,
    use_context=True
)
input_creator = DKNTextIterator model = DKN(hparams, input_creator)
```

After everything is prepared, the model can be trained.

```
model.fit(train_file, valid_file)
```

In the hyperparameters, there are five epochs defined. In the training process, it can be seen that the score in the validation set increases.

```
at epoch 1
eval info:
auc:0.9233, group_auc:0.9227, mean_mrr:0.871, ndcg@2:0.8764, ndcg@4:0.9031, ndcg@6:0.9044
at epoch 2

eval info: auc:0.9389, group_auc:0.9359, mean_mrr:0.8922, ndcg@2:0.8978, ndcg@4:0.9189, ndcg@6:0.9201
at epoch 3
eval info: auc:0.9449, group_auc:0.941, mean_mrr:0.8986, ndcg@2:0.905, ndcg@4:0.9241, ndcg@6:0.9249
at epoch 4
eval info: auc:0.9483, group_auc:0.9457, mean_mrr:0.906, ndcg@2:0.9126, ndcg@4:0.9298, ndcg@6:0.9305
at epoch 5

eval info: auc:0.9496, group_auc:0.9481, mean_mrr:0.9091, ndcg@2:0.9168, ndcg@4:0.9321, ndcg@6:0.9328
```

In the last, the model can be tested on the testing set, and the results are shown below.

```
model.run_eval(test_file)
```

```
{'auc': 0.94, 'group_auc': 0.9374, 'mean_mrr': 0.7071, 'ndcg@2': 0.6735, 'ndcg@4': 0.746, 'ndcg@6': 0.7
```

6.3.3 Evaluation Metrics and Methods

Evaluation is the key step in the application of the machine learning algorithm. For the recommender system, evaluation is very important and it can be even more sophisticated compared to the conventional machine learning system. The reason for this is that in addition to the common evaluation, the recommender system needs to be evaluated online as well, to make sure that in the actual application the recommendation results meet the expectation [6]. The following content of this chapter introduces the commonly seen evaluation metrics and methods.

There are mainly two groups of evaluation metrics for recommender systems. One is to evaluate the performance of the recommender model, and another is to evaluate the business impact of the recommender system. Correspondingly, the methods that are used in these two different scenarios are called offline evaluation and online evaluation.

The offline evaluation, as name suggests, is the ones that are usually used for evaluating the performances of the models that have not been alive. As aforementioned, these metrics are primarily meant for the intrinsic characteristics of the models themselves. The commonly seen metrics can be classified based on the detailed aspect they evaluate on the model. For those models that generate scores, many times the rating-based metrics are effective. These metrics include Root Mean Squared Error (RMSE), coefficient of determination (R^2), mean average error (MAE), and explainable variance. For the sake of saving space, the detailed formulas that calculate these metrics are not introduced here. The MovieLens dataset as discussed before (noted as df_true as a dataframe in the code examples) is used to demonstrate the metrics. In the example below, the data consists of three different dimensions which are user id, item id, and user ratings. To verify the rating-based metrics, there is another prediction dataset, df_pred, where the predicted ratings of users are presented. The codes that prepare the data are shown below.

```
df_true = pd.DataFrame(
    {
        UserId: [1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
    3, 3, 3],
        MovieId: [1, 2, 3, 1, 4, 5, 6, 7, 2, 5, 6, 8, 9, 10, 11, 12, 13,
    14],
        Rating: [5, 4, 3, 5, 5, 3, 3, 1, 5, 5, 5, 4, 4, 3, 3, 3, 2, 1],
    }
)

df_pred = pd.DataFrame(
    {
        UserId: [1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
    3, 3, 3, 3],
        MovieId: [3, 10, 12, 10, 3, 5, 11, 13, 4, 10, 7, 13, 1, 3, 5,
    2, 11, 14],
        Prediction:
        [14, 13, 12, 14, 13, 12, 11, 10, 14, 13, 12, 11, 10,
    9, 8, 7, 6, 5]
    }
)
```

There are several functions available in Microsoft Recommenders for calculating the rating-based metrics. The prerequisites for using these functions are that both the ground truth and the prediction data should contain the user ID, item ID, and users ratings (both the predicted ones and the actual ones). The function takes the two datasets as input and check the items that each user has rated against those that are in the prediction data. The comparison is averaged to generate the final rating-based metrics. The following codes illustrate how the functions are used for computing the rating-based metrics. The readers can calculate manually and compare the results with the ones that are generated from the functions.

```
from recommenders.evaluation.python_evaluation import rmse, rsquared, mae,
exp_var

# Calculate RMSE
rmse(df_true, df_pred, col_user=UserId, col_item=MovieId, col_rating=Rating, col_prediction=Predictior

# Calculate MAE
mae(df_true, df_pred, col_user=UserId, col_item=MovieId, col_rating=Rating, col_prediction=Prediction)

# Calculate R Squared

rsquared(df_true, df_pred, col_user=UserId, col_item=MovieId, col_rating=Rating, col_prediction=Predict

# Calculate explained variance

exp_var(df_true, df_pred, col_user=UserId, col_item=MovieId, col_rating=Rating, col_prediction=Predicti
```

There are usually no absolute metrics to evaluate recommender systems that can hold forever. However, in general, these metrics can be effectively used for comparing two different recommender system implementations.

There is another group of evaluation metrics, which are termed as ranking metrics. These metrics are mainly for assessing "whether the recommendation results are relevant to users' preferences or not." The way of analyzing the relevancy is that, after generating the recommendation results, check whether the users show interest to the recommended items, and, if the users have interests it means the recommendation is "relevant." In some applications, the order of the recommendation results matter as well. For example, if the top-ranked items in the recommendation results have gained users' interest, then the recommendation model that produced such results outperforms, in terms of ranking metrics, those models where lower-ranked items have gained interest. There are several commonly seen metrics, precision (precision@k), recall (recall@k), Mean Average Precision, Normalized Discounted Cumulative Gain (NDCG@k). It is worth mentioning that the use of the constant "k" that is used in these metrics is because that the ranking metrics should consider the number of the recommended items, i.e., k, which affects the evaluation metrics. To comparatively study different recommender models by using these metrics, the constant of k needs to be fixed.

Similar to the rating metrics, there are a few utility functions available in Microsoft Recommenders for calculating the ranking metrics. These functions also require the input datasets of the ground truth and the prediction. The contents of the input datasets are the same as those in the case of the rating metrics calculation functions. If there are ratings in the two input datasets, the rating column does not have to be ranked before the data are used in the function for calculating the metrics. The internal implementation will do the ranking based on the input argument. For example, for the most popular ranking operation (it is called "topk" ranking in Microsoft Recommenders), the function ranks the actual ratings and the predicted ratings of the same users and then use the ranked results for calculating the metrics. The results are shown in Table 6.1. The example below illustrates how the functions are used in the MovieLens dataset for calculating the ranking metrics.

Table 6.1 Comparison between the evaluation metrics

Metrics	Value range	Criteria of model selection	Limitation
RMSE	> 0	The smaller the better	Compared to MSE there may be bias, and the explainability is poor
R2	≤ 1	The closer to 1 the better	It depends on the distribution of the variables
MAE	≥ 0	The smaller the better	It depends on the value range of the variables
Explained Variance	≤ 1	The closer to 1 the better	It depends on the distribution of the variables

```
from recommenders.evaluation.python_evaluation import precision_at_k,
recall_at_k, ndcg_at_k, map_at_k

# Compute precision@10
precision_at_k(
    df_true,
    df_pred,
    col_user="UserId",
```

```
    col_item="MovieId",
    col_rating="Rating",
    col_prediction="Prediction",
    relevancy_method= "top_k",
    k=10
)

# Compute recall@10
recall_at_k(
    df_true,
    df_pred,
    col_user="UserId",
    col_item="MovieId",
    col_rating="Rating",
    col_prediction="Prediction",
    relevancy_method= "top_k",
    k=10
)

# Compute ndcg@k
ndcg_at_k(
    df_true,
    df_pred,
    col_user="UserId",
    col_item="MovieId",
    col_rating="Rating",
    col_prediction="Prediction",
    relevancy_method= "top_k",
    k=10
)
# Compute MAP
map_at_k(
    df_true,
    df_pred,
    col_user="UserId",
    col_item="MovieId",
    col_rating="Rating",
    col_prediction="Prediction",
    relevancy_method= "top_k",
    k=10
)
```

Similar to the rating metrics, there are few tips for the ranking metrics which are summarized in Table 6.2.

Table 6.2 Comparison between the ranking metrics

Metrics	Value range	Selection criteria	Limitation
precision@k	$\geq 0 \& \leq 1$	The closer to 1 the better	It can only be used for evaluating the relevancy of the items in the recommendation results.
recall@k	$\geq 0 \& \leq 1$	The closer to 1 the better	It can only be used for evaluating the relevancy of the items in the ground truth set.
ndcg@k	$\geq 0 \& \leq 1$	The closer to 1 the better	NDCG does not check the missing items, and it cannot evaluate the items that are ranked at the same position.
MAP	$\geq 0 \& \leq 1$	The closer to 1 the better	It depends on the distribution of the variables.

In many application scenarios, the recommendation results are binary, e.g., “like or dislike,” “click or non-click,” “purchase or non-purchase,” etc. The offline evaluation for such situation can leverage the evaluation

methods that are used in the classification problem. Common metrics for such application are AUC, logloss, etc. These metrics check every prediction result and compare it against the real one to see if they are identical. In Table 6.3, the comparison for such binary evaluation methods is summarized. Also, as shown in the code examples, these metrics can be computed by using the functions in Microsoft Recommenders.

Table 6.3 Comparison between AUC and logloss

Metrics	Value range	Selection criteria	Limitation
AUC	$\geq 0 \& \leq 1$	The closer to 1 the better. A value of 0.5 indicates a random guess.	It depends on the size of the recommendation.
logloss	≥ 0	The closer to 0 the better.	It is not stable when the data of the two classes are not balanced.

```
from recommenders.evaluation.python_evaluation import auc, logloss

auc(df_true, df_pred, col_user="UserId", col_item="MovieId", col_rating=
    "Rating")

logloss(df_true, df_pred, col_user="UserId", col_item="MovieId",
col_rating="Rating", col_prediction="Prediction")
```

In addition to the offline evaluation metrics, there are some other metrics which are suitable to some special needs. Such metrics include diversity, serendipity, novelty, etc. These metrics make it possible to design a better recommender system. Due to the limit of the space, the details of these metrics are not introduced. The readers can refer to the [6] for more information.

As aforementioned, a recommender system is an online system and hence the online evaluation is vital. Many engineering and experimenting practices prove that there may be no correlation between the offline and the online evaluations. This does not say that the offline evaluation is not necessary. Many times, the root cause of the discrepancy is owing to the dynamics when the system goes alive (e.g., data drift), and this results in the inefficacy of the deployed model. To avoid the consequence due to the offline-online discrepancy, performing the timely online evaluation is very important. There are many different methods for online evaluation in recommender system. The common ones are A/B testing and multi-armed bandits. The main difference between the offline and the online evaluation is that, in an online setting, the actual feedback from users is collected and used as references to evaluate whether the system performance meets the requirement. The metrics that are used in the online evaluation are usually business oriented. In many circumstances of an e-commerce platform, the recommendation results generated from a system are those that are relevant to the users' interests, and as a result, the recommendation results promote the purchase behavior of the users. The business-related metrics here can be the conversion rate. In the online evaluation, the online evaluating system will assess whether the conversion rate of users is improved after the recommender system is incorporated. The verification procedure can be conducted by using the A/B testing or multi-armed bandit framework.

As the name suggests, A/B testing splits the traffic online into the experiment group (group A) and the control group (group B), and within the limited period of time, the evaluation is performed regarding a particular metric and the statistical testing is then conducted to assess whether there is a change in the two groups. In the recommender system, the experiment group consist of the users who are exposed to the recommendation results whilst those in the control group are not. It is mentioning that, in the traffic split, the two groups need to be isolated carefully, such that there is no information leakage among the two groups. One of the biggest challenges in A/B testing is that, by virtue of a statistical method, it requires sufficient large size of the samples to make reliable conclusion from the experiment. This leads to a strong dependency on the time to conduct the experiment, which may become a bottleneck for the iteration of the recommender system and thus creates deficiency in refreshing the recommender model in the system.

The method of multi-armed bandit originated from the slot machine in the casino - the gambler needs to choose whether to bet on a single machine or try luck on a different in order to optimize the overall gain. A multi-armed bandit usually models the gain probability for each of the options and dynamically tunes the model parameters to maximize the overall gain. The dynamic tuning is performed based on the observations from the online feedback. The commonly used methods in the multi-armed bandit evaluation framework are the Upper Confidence Bound method, Thompson sampling, etc. The gain of each "arm" in the setup is defined based on the business metrics such as click-through rate, conversion rate, etc. The idea of the multi-armed bandit is the same as the recommender

system in terms of the philosophy of “exploration or exploitation.” Compared to A/B testing, multi-armed bandit leverages the dynamics of the online system such that the feedback to one or multiple recommender systems can be effectively captured to assess whether there is a time point where the overall gain is optimized. The characteristics of the multi-armed bandit method makes it independent from the statistical testing so that it saves the time for conducting a sound statistical experiment. In addition, the multi-armed bandit inherits the concept and methods from reinforcement learning. In some cases, it can work independently even without the offline machine learning model (Azure Personalizer in the Microsoft Azure cloud platform makes use of the contextual information to do the personalized recommendation for the items online).

Given that the code base of Microsoft Recommenders does not contain the online evaluation related code examples, the practical examples are not introduced here. The readers can refer to the online learning materials from the projects such as Vowpal Wabbit, Azure Personalizer, Ray, etc. for further reading.

6.4 Development and Operation of Recommender Systems in the Cloud

6.4.1 Advantages of Using Cloud for Recommender Systems

Today, it is not easy to build a high-performance and scalable recommender system due to the explosive growth of data. As discussed in 6.2.3, it takes a lot of time and effort to build an industry-grade recommender system from scratch. The rapid maturing of cloud computing technologies can greatly simplify the buildout of recommender systems. Using cloud computing has the following advantages when it comes to recommender systems.

First, cloud platforms provide powerful and elastic computing resources. When it comes to large-scale recommender systems, scalability is a key design consideration. Training and scoring of large-scale recommender models often require a lot of computing resources. In addition, the online data processing scale of recommender systems may change over time, which makes the recommender system’s demand for computing fluctuate. Therefore, it is very important to have computing resources that can easily adapt to these changes. Cloud platforms can provide very reliable and convenient services for both of these requirements. Developers can easily set the allocation of resources in their scheduling programs to minimize the cost of computing while ensuring the efficiency of computing. At the same time, many cloud platforms have already helped some recommender models to be trained on large clusters, which greatly improves the efficiency of model development.

Second, in addition to computing power, cloud platforms also provide a rich set of data stores. For complex recommender systems, it is important to having access to a diverse collection of data store interfaces. This is because various data stores can play distinct roles at different stages of the data pipeline. For examples, large-scale data stores that support unstructured data are often needed during data preparation, while distributed document databases are valuable when retrieving recommended items.

Third, cloud-based services are often easier to deploy and operate. Compared to starting from scratch in an on-premise environment, developers can readily use the full range of services provided by a cloud platform, making it easy to develop recommender systems end-to-end. For example, traditionally, when developing a recommender system requires considering both scalability of the backend systems and latency of the front-end systems. These requirements are often complex and time-consuming to implement. By using the ready-made cloud services, developer teams can minimize or avoid such complexities significantly reducing development time.

Fourth, cloud platforms often provide a good environment for developing, experimenting, and deploying recommender systems. For algorithm engineers, It is productive to have a machine learning development environment that is not only pre-configured with common models and toolkits but also well-integrated with the rest of the cloud platform for deployment. Many cloud platforms have such integrated machine learning environment, including Azure Machine Learning Service on Microsoft Azure. With such integrated environment, algorithm engineers can focus on developing models and be productive in integration and deployment.

Fifth, cloud platforms typically have built-in security and reliability features important to production-grade recommender systems. These features may cover diverse aspects such as network, data, and service and often require significant amount of resources to implement.

6.4.2 Cloud-Based Development and Operations of Recommender Systems

Compared to the traditional development and deployment, building recommender systems on cloud platforms can take advantage of existing services. A big part of work for development teams is to integrate these services into a production-grade recommender system. In this section, we will use an example of building a real-time recommender system based on collaborative filtering on Microsoft Azure cloud platform to illustrate how to

develop and operate a recommender system on cloud platforms. The details of this example can be found in the code repository of Microsoft Recommenders, at examples/05_operationalize/als_movie_o16n.ipynb.

Figure 6.5 shows the architecture of this recommender system. In this example, we continue to use the MovieLens dataset. In practice, this kind of recommender system could be used, for example, in an online video entertainment platform. The system recommends videos to users based on their preferences to improve user experience.

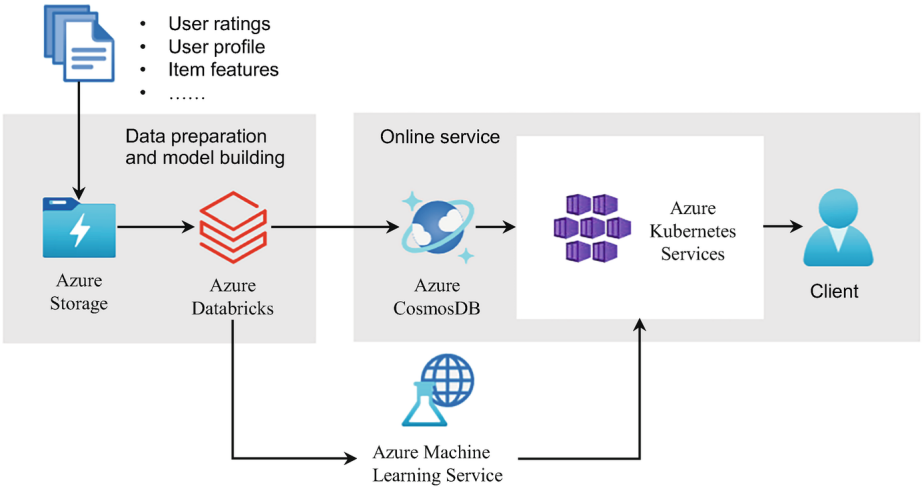


Fig. 6.5 The architecture diagram of the Azure cloud-based real-time recommender system

The dataflow of this recommender system is as follows:

1. The video entertainment platform collects the viewing history of its users in accordance to relevant privacy requirements. In particular, the data used to represent the user's interest in the video content, such as the clicks and ratings, will be used to train the recommender system.

2. The collected data is stored in a cloud storage service. In this example, we use Azure Storage Account. This service does not require a specific data format, can store and read massive amounts of data, and also encrypts data for security.

3. The data is loaded into Azure Databricks for data processing and model training. Azure Databricks is a distributed computing service on Azure that supports computing environments such as Spark. Data scientists and algorithm engineers can write and run code in notebooks on Azure Databricks without having to set up or maintain the underlying Spark environment or cluster.

4. After loaded into the environment in Azure Databricks, the data goes through preprocessing and is split into training and test sets. The processed data is used to train the recommender model. For the sake of simplicity, in the example in Microsoft Recommenders, we use the ALS algorithm that comes with Spark. The ALS algorithm is a matrix factorization algorithm. The Spark implementation of ALS uses Spark's distributed computing mechanism to make the least squares iteration in the model training process efficient.

5. After the model is trained, we can evaluate the model using metrics discussed in 6.3.3, such as ranking metrics and rating metrics.

6. We can use the trained model to recommend products to users who appear in the training data. Since the collaborative filtering model can generate recommendations immediately after the model is trained, the batch-based architecture can easily store the recommendations in a high-performance database so that the recommendations can be retrieved in real time. In this architecture, we use Azure Cosmos DB, a distributed database service on Azure. It is a NoSQL database supporting different data formats, but it also supports SQL and other types of data interfaces. In addition, Azure Cosmos DB is globally distributed, allowing recommendation output to be available in different regions for the recommendation service.

A Trained model is deployed to the Azure Kubernetes Service API server using Azure Machine Learning Service, so that the recommendation results can be easily accessed and used by various applications. In the special case of the collaborative filtering model, recommendations can be generated immediately and stored in Azure Cosmos DB. The API deployed by Azure Machine Learning Service does not need to call the model to score based on the request; instead, the API can directly query Azure Cosmos DB before serving.

7. When the video entertainment platform client needs to be shown a recommended video, the request is sent to the API on Azure Kubernetes Service via HTTPS. The logic code deployed on Azure Kubernetes Service will then search for the relevant content in the recommendations stored in Azure Cosmos DB based on the parameters in the API, such as the user ID, and return the result to be displayed on the user interface.

In this example architecture, Azure Databricks, Azure Cosmos DB, and Azure Kubernetes Service are used for recommendation. To meet the performance requirements of the recommender system, stress tests can be performed to tune the system configuration and size. As for scalability, it is important to be able to adjust the system scale to meet changing usage volume. Azure Databricks cluster size (the number of compute nodes) can be adjusted easily during operations. It also supports auto-scaling to optimize the utilization of computing resources. In the serving side, the size of Azure Kubernetes Service cluster and the throughput of Azure Cosmos DB can both be adjusted to improve operational efficiency.

6.5 Summary

This chapter focuses on some problems and considerations in industry applications of recommender systems and discusses the details of these actual applications based on the code in the Microsoft Recommenders repository and an cloud-based reference architecture. Readers are encouraged to follow the steps and methods in the text in a hands-on manner and experiment using the algorithms introduced in previous chapters.

This chapter is not intended to be a comprehensive overview of industry-grade recommender system research and practice. Up-to-date industry research results can often be found in leading conferences. The industrial paper section of ACM SIGKDD and the application papers and example presentations of ACM RecSys are good places to start.

References

1. Xavier Amatriain and Justin Basilico. “Recommender Systems in Industry: A Netflix Case Study”. In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. Boston, MA: Springer US, 2015, pp. 385–419. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6_11.
2. *Apache Spark*.URL: https://spark.apache.org.
3. Andreas Argyriou, Miguel González-Fierro, and Le Zhang. “Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems”. In: *Companion Proceedings of the Web Conference2020*. WWW ’20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 50–51. ISBN: 9781450370240. DOI: 10.1145/3366424. 3382692.
4. Shafi Bashar and Alex Gillmor. *Scaling Collaborative Filtering with PySpark*. Yelp Engineering Blog. May 7, 2018. URL: https://engineeringblog.yelp.com/2018/05/scaling-collaborative-filtering-with-pyspark.html.
5. Antoine Bordes et al. “Translating Embeddings for Modeling Multirelational Data”. In: *Advances in Neural Information Processing Systems 26. 27th Annual Conference on Neural Information Processing Systems 2013*. Ed. by C.J. Burges et al. Curran Associates, Inc., 2013, pp. 2799–2807. URL: https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.
6. Asela Gunawardana and Guy Shani. “A Survey of Accuracy Evaluation Metrics of Recommendation Tasks”. In: *Journal of Machine Learning Research* 10.100 (2009), pp. 2935–2962. URL: http://jmlr.org/papers/v10/gunawardana09a.html.
7. Larry Hardesty. *The history of Amazon’s recommendation algorithm. Collaborative filtering and beyond*. Amazon Science. Nov. 22, 2019. URL: https://www.amazon.science/the-history-of-amazonsrecommendation-algorithm.
8. Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative Filtering for Implicit Feedback Datasets”. In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 263–272. DOI: 10.1109/ICDM. 2008.22.
9. Vinh Nguyen et al. *Accelerating Recommender Systems Training with NVIDIA Merlin Open Beta*. NVIDIA Developer. Oct. 5, 2020. URL: https://developer.nvidia.com/blog/accelerating-recommendersystems-training-with-nvidia-merlin-open-beta/.
10. Shumpei Okura et al. “Embedding-Based News Recommendation for Millions of Users”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1933–1942. ISBN: 9781450348874. DOI: 10.1145/3097983.3098108.

11. Qi Pi et al. “Practice on Long Sequential User Behavior Modeling for Click-Through Rate Prediction”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2671–2679. ISBN: 9781450362016. DOI: 10.1145/3292500.3330666.
12. Guorui Zhou et al. “Deep Interest Evolution Network for Click-Through Rate Prediction”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 – February 1, 2019*. AAAI Press, 2019, pp. 5941–5948. DOI: 10.1609/aaai.v33i01.33015941.

D. Li et al., *Recommender Systems*
https://doi.org/10.1007/978-981-99-8964-5_7

7. Summary and Outlook

Dongsheng Li[1] ✉, Jianxun Lian[2], Le Zhang[3], Kan Ren[4], Tun Lu[5], Tao Wu[6] and Xing Xie[2]
(1) Microsoft Research Asia, Shanghai, China
(2) Microsoft Research Asia, Beijing, China
(3) Standard Chartered (Singapore), Singapore, Singapore
(4) ShanghaiTech University, Shanghai, China
(5) School of Computer Science, Fudan University, Shanghai, China
(6) Microsoft, Cambridge, MA, USA

Abstract
This chapter provides a summary of the book and offers insights into future trends in the research and application of recommender systems.

Keywords Summary – Future trends of research and application

In the information age, recommender system has become an indispensable application, helping people obtain information more efficiently in various fields, such as e-commerce, movies, books, music, and news. In recommender systems, the two most important elements are users and items. Around these two elements, researchers in the field of recommender systems have proposed a large number of innovative research works, mainly including two categories: content-based recommendation algorithms and collaborative filtering-based recommendation algorithms. The content-based recommendation algorithm mainly studies how to model and describe structured content and unstructured content and then recommend items similar to user interests to users. The collaborative filtering algorithm mainly studies how to find neighbors with similar interests for target users, and how to recommend items for target users based on the interests of

neighbors. These two ideas are still the basis of recommendation algorithm research nowadays.

With the continuous development of deep learning technology, mainstream recommendation algorithms are gradually replaced by deep learning technology. For example, the traditional description of content is replaced by representation learning that deep learning is good at, such as the vector space model of text modeling has been replaced by neural network models. In addition, according to the characteristics of neural networks, researchers have explored a new paradigm of recommendation algorithms, that is, representation learning plus interaction function learning. Among them, representation learning mainly expresses users and items as vectors through neural networks, and interaction function learning mainly learns the relationship between user vectors and item vectors through neural networks. This new paradigm takes advantage of deep learning to model various types of information more flexibly, such as user characteristics, item attributes, sequential information, and graph information as well as more complex relationships between users and items, such as high-order non-linear relationship, etc. Deep learning technology has dominated the research and applications related to recommender systems, and it is believed that this trend will continue until the emergence of alternative technologies for deep learning.

As recommender systems gradually penetrate into more scenes of people's daily life, the interaction between recommender systems and people is also facing greater challenges. As an application that is closely connected with people, the bottom line of recommender systems is not to harm users, that is, it complies with the relevant guidelines of responsible artificial intelligence. For this goal, researchers need to pay attention to the security and privacy of recommender systems, the interpretability of the algorithm, whether the algorithm is biased, etc. In addition to the impact on individuals, it is also necessary to pay attention to the impact of recommender systems on social groups, such as whether a recommender system will produce an information cocoon effect, etc. Therefore, while paying attention to technological development, we must also pay attention to the possible negative social impact of recommender systems. When researching new technologies, we should try our best to ensure that the technology is responsible. This point has been recognized by mainstream

research institutions around the world and will be the focus of the development of recommender system-related technologies in the future.

Recommender systems are closely related to applications, so when learning the technology related to recommender systems, it is necessary to combine theoretical knowledge with practical experience. To this end, this book introduces the practical experience of recommender systems based on Microsoft Recommenders, an open source project of Microsoft. Readers can learn more deeply about the design principles and practical methods of recommendation algorithms based on the source code provided in this book and can quickly build an accurate and efficient commercial recommender system from scratch.

“Long, long had been my road and far, far was the journey; I will go up and down to seek my heart’s desire.” There are still many key problems that need to be solved in the seemingly mature field of recommender systems, such as causality, common sense, etc. It is believed that researchers and engineers will be able to solve them in the near future, bringing a new wave of recommender system research and application.