Reza Ravanmehr
Rezvan Mohamadrezaei

Session-Based Recommender Systems Using Deep Learning

Springer

Session-Based Recommender Systems Using Deep Learning

Reza Ravanmehr • Rezvan Mohamadrezaei

Session-Based Recommender Systems Using Deep Learning

Springer

Reza Ravanmehr
Department of Computer Engineering,
Central Tehran Branch
Islamic Azad University
Tehran, Iran

Rezvan Mohamadrezaei
Department of Computer Engineering,
Central Tehran Branch
Islamic Azad University
Tehran, Iran

ISBN 978-3-031-42558-5 ISBN 978-3-031-42559-2 (eBook)
https://doi.org/10.1007/978-3-031-42559-2

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Dedicated to
our families for their support
during the preparation of this book
and to
all our students for their ideas,
motivations, and patience.

Preface

The Web has become an essential part of human life, making it possible for everyone everywhere to share information, access opportunities, and collaborate across geographic and cultural limitations. Millions of data objects are uploaded to the Web every minute, and the amount of data is increasing exponentially. It is difficult and even impossible for users to study all available data and discover the required and favorite items among them. The recommender system is an important tool to facilitate users' decision-making process. These systems utilize the users' short- and long-term data in addition to the results of other users' preferences and decisions to predict the favorite items of the target user.

The role of recommender systems in the real world is vital and undeniable, considering the increasing different types of service delivery and customer satisfaction in e-commerce, multimedia and entertainment, jobs, news, etc. Recommender systems are essential to increase user satisfaction and predict their favorite items, but there are also various challenges in this field. The traditional recommender systems rely on user profiles to generate personalized recommendations, but sometimes user profiles do not exist or user authentication policies prevent access in real-life applications. Indeed, the privacy of users' personal information is becoming a big challenge, and therefore, recommendation systems should reduce the dependence on user profiles without adversely affecting the accuracy of recommendations. It should be mentioned that despite permitting access to users' long-term interests, the users may still need an item that can be predicted using only their short-term data.

The solution is to predict and provide recommendations based on user session interactions. A session is a list of user interactions considered in specific time intervals. A session for a particular user usually consists of a set of interactions related to the user that occurred at different times in the session. There are two types of sessions: ordered and unordered sessions. An ordered session refers to a session in which the interactions are chronologically ordered, but unordered sessions include a set of interactions that are not in any particular order, such as purchases made by a customer. Items in a session contain metadata such as name, description, and related categories, which makes it easier to predict the user's favorite items.

A session-based recommender system (SBRS) provides recommendations employing session data without the need to access long-term user data and interactions. Due to the availability of required data for session-based recommender systems and the compatibility of the features of these systems with real-world applications, SBRS has attracted the attention of many researchers. Of course, it should be noted that there are very narrow boundaries between session-based recommender systems and sequential recommender systems. Session-based recommender systems are modeled on the basis of session data, and sequential recommender systems are modeled on chronologically sequential data. There is a difference between a sequence, which is a list of previous time-ordered interactions without defined time intervals, and a session, which consists of a list of interactions on different items with a defined boundary over a relatively short period.

The purpose of session-based recommender systems is to predict one or a group of items of a session or to predict the next session based on learning the dependencies within each session or between several sessions. These dependencies are recognized based on the simultaneous occurrence of the last few user interactions. On the other hand, in sequential recommender systems, items are predicted based on learning sequential dependencies between consecutive items in different sessions. In fact, sequential recommender systems capture users' long-term preferences across previous sessions as conventional recommendation systems do, but they also need to model users' short-term interests in a short sequence.

Various methods have been proposed to model SBRS, such as association rules, Markov chains, pattern mining, latent representation, and deep neural networks. Each of these methods has been used in different research. Deep neural networks are approaches that have received much attention due to their high performance and capability to model complex problems. These techniques have been widely used in many academic and commercial fields recently. In session-based recommender systems, various deep neural network techniques are used to identify dependencies between the interactions of a session and complex relations between different sessions. Researchers have been more interested in developing session-based recommender system based on deep learning techniques in the last few years. This fact proves the effective and undeniable role of deep learning techniques in session-based recommender systems. The session-based recommender system using deep learning approaches is mainly divided into basic and hybrid/advanced deep neural network models. Basic deep neural network approaches include RNNs, CNNs, MLPs, AEs, GANs, etc., while hybrid/advanced deep neural networks include GNNs, attention-based models, deep reinforcement learning, and models obtained from the specific combination of basic methods.

Aims and Scope

This book focuses on the widespread use of deep neural networks and their various techniques in session-based recommender systems. In fact, the authors intend to draw the audience's attention to the fact that the process of choosing the appropriate deep learning technique for recommender systems whose input data are sessions should be done according to the type and context of session data and the goals of the problem. In this book, an attempt has been made to present the success of using deep learning techniques in many research on the session-based recommender system from different perspectives. For this purpose, the concepts and fundamentals of session-based recommender systems are fully elaborated. Then, different deep learning techniques, specially focusing on the main subject of the book, are studied. A review of studies in session-based recommender systems shows that various deep learning approaches, such as discriminative, generative, and graph-based models, have been utilized to provide effective and accurate recommendations. In addition, we review and discuss different learning to rank (LtR) methods which are inevitable parts of a session-based recommender system to achieve a better recommendation quality and improve the ranking performance.

Main Emphasis

The main emphasis of this book is to deliver a precise, comprehensive, and up-to-date perspective of the concepts, challenges, types, architectural details, evaluation methodologies, and pros/cons of the methods presented in the session-based recommender system using deep learning models. To this end, technical descriptions, highlights, limitations, and strengths/weaknesses of different deep neural network models in session-based recommender systems are discussed and analyzed.

Target Audience

According to the materials presented in the different chapters of the book, the authors try to give the audiences of the book a comprehensive view and sufficient information to elaborate and develop session-based recommender systems. In addition, the contents of this book are prepared so that the readers can learn the fundamental concepts related to session-based recommender systems and deep learning separately and provide them with the details of the deep learning approaches applied in the session-based recommender systems.

In fact, this book is provided for researchers who intend to use deep learning models to solve the challenges related to session-based recommender systems. In brief, the main target group can be classified into the academy (graduated students)

and the industry (RS applications/product developers and designers). The material covered in the book addresses the areas of recommender systems, Web data mining, information retrieval, and machine/deep learning.

Prerequisites

We assume that all readers are familiar with computer science concepts. To better understand the contents of the book, readers should be familiar with the basic concepts and principles of designing and analyzing algorithms and machine learning methods and, to some extent, be familiar with the principles of implementing systems using at least one programming language, preferably Python. Mathematically, we expect that the reader is familiar with calculus, probability theory, and linear algebra.

Short Summary

The book is well modularized, and each chapter can be learned in a stand-alone manner based on individual interests and needs. It is noteworthy to mention that readers familiar with session-based recommender systems can skip Chap. 1 and readers who have already studied the concepts of deep learning methods can skip Chap. 2. In the sequel, a brief introduction to each chapter of the book follows:

In the first chapter of the book, definitions and concepts related to a session-based recommender systems are reviewed. At the beginning of this chapter, we have explained a general overview of recommender systems. In the remainder of the chapter, the main focus of the content is session-based recommender system. For this purpose, we explain each concept more clearly with the related mathematical formulations. Then, the challenges of session-based recommender systems are thoroughly considered. Finally, a taxonomy of SBRS approaches is presented, where the characteristics and applications of each class are discussed separately.

The second chapter starts with the basic concepts of deep learning and the characteristics of each model. Then, each deep learning model, along with its architecture and mathematical foundations, is introduced. For this purpose, deep learning models are classified into discriminative, generative, and graph methods. Deep discriminative models include convolutional neural networks (CNN), recurrent neural networks (RNN), and multilayer perceptron (MLP). The major deep generative models include autoencoders (AE), generative adversarial networks (GAN), and different types of Boltzmann machine (BM) models. Graph-based models also include graph neural networks (GNN) and graph convolutional networks (GCN).

Different approaches of deep discriminative models in a session-based recommender system are discussed and analyzed in Chap. 3. Due to the ability of RNNs to

model dynamic behaviors of session data over time, many approaches in session-based recommender systems employ RNNs. Therefore, RNN approaches (including GRU and LSTM) significantly impact the performance of SBRS. However, because of the capability of CNNs to extract and learn temporal and spatial patterns, session-based recommender systems also use CNNs. MLP-based approaches also are mainly suitable for unordered session data due to their inability to model sequential data.

In the fourth chapter, session-based recommender systems that benefit from deep generative neural networks are discussed. The study and analysis of published research show that deep discriminative methods have broader applications than deep generative methods in session-based recommender systems due to their sequential approach. However, generative techniques are also important for researchers due to their ability to reduce the problems of sparsity and complexity of user interactions in the field of SBRS. This chapter focuses on autoencoder, generative adversarial networks, and flow-based models (normalizing flow and autoregressive flow) in session-based recommender systems.

Chapter 5 discusses session-based recommender systems using hybrid/advanced deep learning methods. Each deep learning method has specific features and capabilities, and due to the high flexibility of deep neural networks, many neural structure blocks can be integrated to formulate more robust and accurate models. The integration of two or more different models provides the possibility of using the advantages of each method, limiting their disadvantages, and strengthening the capabilities of the resulting combined method. Usually, these approaches consist of several stages, in which a deep learning technique is used in each stage to process and generate the required data for the next stage. Therefore, the optimum combination and configuration of these stages in different research are discussed in this chapter. In session-based recommender systems, the combination of CNNs and AEs, CNNs and RNNs, RNNs and AEs, and RNNs and deep reinforcement learning (DRL) has received the most attention. Due to the high efficiency of deep reinforcement learning and graph neural network methods, a significant number of session-based recommender systems are based on these methods. Therefore, Chap. 5 ends with details of various session-based recommender systems using graph neural networks or deep reinforcement learning.

Learning to rank (LtR) methods are based on machine learning techniques in ranking results in different domains. Chapter 6 reviews different LtR methods focusing on information retrieval and recommender system domains. For this purpose, various approaches to rank creation and rank aggregation are discussed and reviewed. This chapter mainly focuses on four approaches to ranking creation: pointwise, pairwise, listwise, and hybrid.

Finally, the results of the investigations and findings from the research review conducted throughout the book are presented in conclusion.

Tehran, Iran

Reza Ravanmehr
Rezvan Mohamadrezaei

Acknowledgements

Writing acknowledgments at the beginning of a book is always the most beautiful part! This book would not have been possible without the contribution and help of many people. First of all, we would like to express our grateful thanks to all of the previous and current faculty members and students of the Department of Computer Engineering at CTBIAU and many friends and colleagues whose constant support and encouragement have made our work on this stage. Appreciation also goes to the friends and colleagues in the Institute for Research in Fundamental Sciences (IPM) who helped make this project a success.

We want to thank Springer staff, in particular Ralf Gerstner, for their dedication and helpfulness throughout the project. We also appreciate the invaluable feedback from all of the reviewers. In addition, we also want to thank those who allowed us to reproduce figures from their publications.

During our career, we were extremely privileged and lucky to work with a fantastic group of students whom we would like to thank for their formative role. We always had plenty of fascinating discussions during courses and research projects that motivated us to write this book.

Last, and most of all, we are grateful to our families for their wholehearted support throughout this project and for giving us the strength to finish this book. Reza would like to thank his wife, Paris, without her help and understanding, it would be impossible to accomplish this project. He would also like to express his gratitude to his parents for their support at different stages of life. Rezvan would like to express her sincere appreciation to her husband, Vahid, for his kindness and motivation while writing this book. And, she would also like to appreciate her parents and brother for their encouragement and support that kept her constantly moving forward.

Contents

About the Authors

Reza Ravanmehr graduated in computer engineering from Shahid Beheshti University, Tehran, in 1996. After that, he gained his M.Sc. and Ph.D. in computer engineering, from SRBIAU, Tehran, in 1999 and 2004, respectively. His main research interests are recommender systems, big data analytics, and social networks. He has published over 60 scientific papers in different computer science disciplines, mainly in social network analysis and recommender systems. He has been a faculty member of the Department of Computer Engineering at Central Tehran Branch, Islamic Azad University, since 2001. E-mail: r.ravanmehr@iauctb.ac.ir

Rezvan Mohamadrezaei is currently a Ph.D. candidate in software systems at Central Tehran Branch, Islamic Azad University. Before that, she received a B.-Sc. degree in computer engineering from MHRIAU, in 2007. After that, she gained her M.Sc. degree in computer engineering from SRBIAU, Khouzestan, in 2012. Her current research interests are in the areas of deep learning, recommender systems, and information retrieval. She has been a faculty member of the Computer Engineering Department at Karoon Institute of Higher Education, Ahvaz, since 2013.

Chapter 1
Introduction to Session-Based Recommender Systems

Abstract Due to the massive growth of data in recent years, recommender systems have become essential tools to improve people's lives. However, access to users' profiles and their long-term interests are crucial challenges of these systems. A session-based recommender system (SBRS) was developed to solve these problems and received much attention from the research community. In this chapter of the book, after presenting an overview of the definitions and techniques of traditional recommender systems, we focus on the fundamental concepts, descriptions, challenges, and approaches of SBRS and clarify the differences between SBRS and SRS (sequential recommender system) from various aspects.

Keywords Session-based recommender systems · SBRS · Sequential recommender systems · SRS · Deep learning

1.1 Introduction

Recommender systems help users make efficient decisions in various fields. The process of selecting favorable items from numerous available items is an ambiguous process, and recommender systems facilitate this process by using data related to the user and the results of other users' decisions. Recommendation systems are divided into different classes based on the type of related processes and the data used for making recommendations. The use of recommender systems is needed to increase user satisfaction and predict the interesting items of users in different fields. However, it has specific challenges, and so far, different types of recommender systems have been proposed to solve these challenges.

Access to users' profiles and their long-term interests are among these crucial challenges. Many users do not create profiles to use different services in a recommendation platform or maybe new users and have performed very limited transactions. In this case, it is very difficult for the systems to decide on their favorite items. On the other hand, even with access to the long-term interests of users, the user may still consider a problem for which short-term data are required for effective prediction. A session-based recommender system (SBRS) was proposed to improve these problems and received the attention of many researchers.

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2_1

Most of SBRS's scenarios include a sequence of often short user interactions in a session, and in many cases, the long-term interests of users are not available [1]. Session-based recommender system have gained importance, and many studies have been conducted on them recently. One of the reasons session-based recommender systems are valuable is that each session is treated as a unit for organizing data. This approach preserves the nature of transactional data and makes decisions based on dependencies within each session. The SBRS modeling is not performed in other types of recommender systems because SBRS usually divides session data into multiple user-item interaction pairs to fit the data into the models. Moreover, it is easier to access session data for session-based recommender systems than the rating data required by content-based or collaborative filtering recommender systems. As a result, these types of systems are more practical in the field of business and e-commerce.

After the introduction, Sect. 1.2 presents an overview of the concepts and approaches of recommender systems. Section 1.3 focuses on the fundamental concepts, definitions, and challenges of session-based recommender system and the differences between session-based recommender system, sequential recommender system (SRS), and session-aware recommender system (SARS) from various aspects, and the ambiguities related to the boundaries between them will be addressed. Section 1.4 takes a brief look at various approaches related to session-based recommender systems.

1.2 Recommender Systems

Recommender systems attempt to provide users with recommendations based on their actions, behaviors, and interests, which match their preferences and make effective decisions. These systems actively and continuously collect and process different types of explicit or implicit data related to items, users, and previous interactions of users to create accurate recommendations [2]. In other words, recommender systems are developed to estimate the usefulness of an item and predict the value behind its recommendation. The prediction of the valuable items for a specific user differs according to the proposed recommender algorithm. In the past years, recommender systems have made significant progress and have focused on different fields such as movies, music, news, jobs, books, and Web sites. Figure 1.1 shows the general process of a recommender system based on user data, item features, and the interactions between users and items.

One of the most basic types of these filtering algorithms is recommender systems based on demographic information. The filtering algorithm is the process of distinguishing useful items to recommend to users. This filtering algorithm of the recommender system is critical, and it determines the basis of the recommendation systems.

Referring to the classic categories of previous research in recommender system scope [3–5], recommendation techniques are divided into three general categories

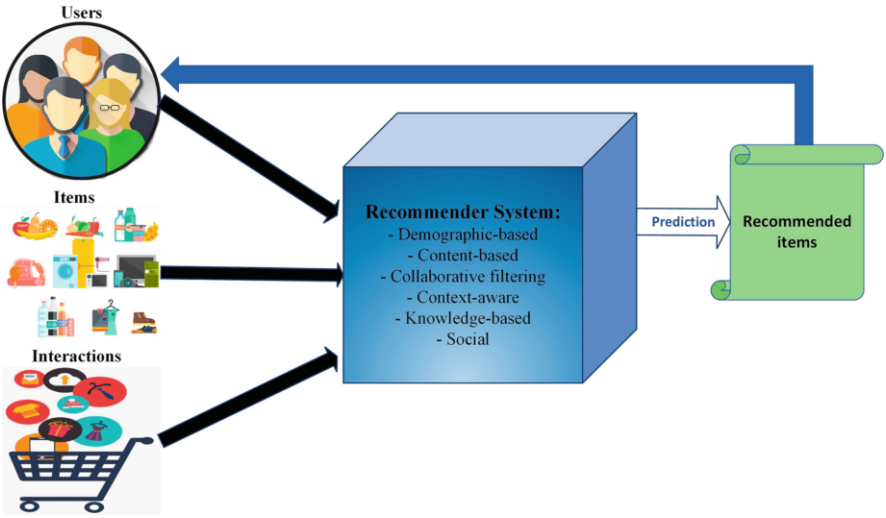

Fig. 1.1 The general process of a recommender system

based on the type of the filtering algorithm: content-based (CB), collaborative filtering (CF), and knowledge-based (KB). Although such categories are still used, many new categories have also been proposed, each with different degrees of overlap: context-aware, utility-based, social, and hybrid recommender systems. Marcuzzo et al. believed a more accurate classification for recommender systems could be provided due to the ever-increasing expansion of data, the big data revolution, and the emergence of data-oriented approaches [6].

Now, we will briefly review the fundamental classes of recommender systems: demographic-based, collaborative filtering, content-based, knowledge-based, context-based, and social recommender systems:

- **Demographic-based recommender systems** consider each user based on his/her demographic information. Demographic information is used to identify the types and categories of users interested in certain items. Demographic-based recommender systems utilize the principle that users with common characteristics (such as the same nationality or the same gender) prefer similar items. Usually, these algorithms are combined with other algorithms, such as content-based or collaborative filtering algorithms, to achieve better results [7].
- **Collaborative filtering recommender systems** utilize the past behaviors and interests of users whose activities are similar to the target user. These types of systems recommend items to each user that have been highly rated by users who are similar to the target user [8, 9]. Collaborative filtering algorithms are divided into memory-based and model-based methods. Model-based methods learn a model based on the ratings that users have previously given for various items, whose learning is implemented based on machine learning or data mining techniques. However, memory-based methods provide recommendations based

on a database created on the relationships between items and users. They are divided into item-based, user-based, and hybrid methods. User-based methods calculate the degree of similarity between users based on the ratings given by users to the same items. Recommending new items or predicting the level of interest in a certain item for a specific user is based on the user's previous interests and the interest of other users similar to him. In item-based methods, recommendations for users are calculated by finding items similar to other items that the user is interested in. Hybrid methods are the result of combining user-based and item-based methods. In memory-based methods, the learning process is done offline, and in addition, all information and data must be in memory for calculations and predictions, which creates a scalability challenge. But model-based methods do not have scalability problems.

- **Content-based recommender systems** focus on item feature analysis. In this approach, the user's profile is built based on the features of the contents of the items that have already been rated by that user [10]. Consequently, items are recommended to each user that is related to the items that are related to his/her previous interest, and these methods work independently of the interests of other users. The recommendations provided by content-based filtering are based on models created on statistical techniques or machine learning methods [2]. The content-based filtering method has the advantage of recommending items that have not yet been rated. If the user's preferences change, this method quickly matches the recommendations with the new preferences. It is not necessary that different users have rated the same items in common so that the amount of their similarity should be determined [7]. In addition to the advantages mentioned above, the disadvantages of the content-based filtering method are the need to access user profiles with complete information about the features of the items to provide effective recommendations to users. Moreover, the recommendations provided in this method are only similar to the items that the user has rated before [2].
- **Knowledge-based recommender systems** are based on explicit knowledge and rules about item domains and user requirements and preferences [11]. A knowledge-based recommender system employs knowledge extracted from the user's previous interactions, in contrast to content and collaborative filtering-based techniques. Knowledge usually includes explicit information related to users and items that are provided by users, and the system creates user profiles using this information. Or, the knowledge used in this type of system is related to a specific domain and context, which is inferred from the usefulness of an item for the user or the adaptation of the features of a particular item to the user's requirements and preferences [7]. Two widely used methods for developing knowledge-based recommender systems are case-based reasoning and constraint-based methods. The case-based reasoning method is an artificial intelligence model that provides reasoning processes for new situations based on the experiences of previous cases of the system [12]. The constraint-based method extracts a set of recommendation rules to find items based on users' requirements [13].

- **Context-aware recommender systems** integrate information sources describing the environment in which interactions occur. Contextual data can include representational contextual data, which are a set of observable contextual variables such as time, place, and weather, or contextual data of interactions that are dynamic and usually include the user's recent activities that are not explicitly visible, such as the current purchase, the user's state, etc. [14]. Another type of contextual data, such as textual comments on an item, is important for recommender systems by new or anonymous users. The most extensive type of contextual information that is studied is temporal [15]. Because user interests change over time, static recommendations may be less effective. Instead, it may be possible to discover patterns in the sequential behavior of users, which is the goal of methods that incorporate time into the recommendation process. Methods in this field usually distinguish between a sequence, which is a list of time-ordered interactions without defined time intervals, and a session, which includes a list of interactions with a defined boundary within a relatively short time interval.
- **Social recommender systems** simultaneously utilize user-to-item interactions as well as user-to-user social relations for the task of generating item recommendations to users [16]. The most prominent types of social recommender systems are the recommendation of social media content and the recommendation of people [17, 18]. The important domains of social recommender systems are mostly related to the content delivered on the Internet and include blogs, multimedia, community question-answering systems, jobs, news, and microblogs. Groups and communities play a crucial role in social media platforms which makes group recommendation techniques highly relevant for the social recommender system domain.

In Table 1.1, the abovementioned recommender system types are compared with each other in terms of input data, basic assumptions, mechanisms, advantages, and disadvantages.

To provide accurate and effective recommendations to users, all kinds of recommender systems face challenges that they must handle. The most important common technical challenges among all types of recommender systems are cold start, delay in providing recommendations, changing user preferences, data sparsity and fragmentation, scalability, security, appropriate user interface design, and selection of appropriate evaluation measures. Various studies have been presented to reduce each of the challenges in different scopes. The readers who are interested in recommender system should refer to a comprehensive handbook dedicated entirely to the field of recommender systems contributed by leading experts in this field [19].

Table 1.1 Comparison of different types of recommender systems

Recommender system	Input	Basic assumption	Mechanism	Advantages	Disadvantages
Demographic-based recommender system	Demographic data of users and items	Users who have similar demographic information are interested in similar items	Classification of users based on demographic information and recommendation of items based on the interests of similar users	No need for information about the user's previous interactions	Users' privacy violation, lack of access to the demographic information of all users, ignoring the change of interest of users
Content-based recommender system	Implicit or explicit content data of users and items	The user is interested in an item that is similar to his/her previous favorite items	Accordance of the user profile data with the content of the items	Simple and understandable, reducing the cold start problem	Only makes recommendations based on the current user's interests
Collaborative filtering recommender system	User-item interactions' matrix	The user is interested in an item that is similar to the items liked by others	Modeling user-item interactions and finding a set of users with similar interests to a particular user	Direct and intuitive to implement, relatively simple mechanism	Data sparsity, cold start, and scalability, high computation cost for large data
Context-aware recommender system	Contextual features and interactional data of users and items	Users have different interests in different contexts and constraints	Modeling item and user interactions based on contextual data	Adaptive recommendations to different conditions	Unavailability and the problem of privacy of contextual data
Knowledge-based recommender system	User's previous interactions and knowledge of specific domains and context	The user's favorite items are obtained based on the knowledge extracted from domain and user-item interactions	Extracting knowledge using methods, such as neural networks, meta-heuristic algorithms, and graphs, and mapping user requirements and item features	Supporting changes in users' interests, suitable for conversational and interactive systems	The difficult process of collecting information and requiring knowledge engineering
Social recommender system	User-item interactions, user-user social relations, and user profiles	Users who establish social relationships have similar interests	Modeling item and user interactions based on social relations	Reducing the cold start problem, supporting changes in users' interests	Privacy, unstable accuracy

1.3 Fundamentals of Session-Based Recommender Systems

Many conventional recommender systems have several fundamental challenges [20]. One of the crucial challenges is their focus on the long-term interests of users statistically, which means that the patterns of short-term interests of users are ignored. As a result, the change in users' interests over time is not taken into account, and the specific needs and items desired by the user in a certain period of time may be affected by his long-term interests [21]. To make effective recommendations to the user, recommender systems divide a basic transaction unit (such as a session) into several records with smaller granularity (such as a user-item interaction) during data processing. This process destroys the sequential nature of the interactive behavior of users, which shows changes in their behavior and interests.

Another problem of the recommender system is the non-availability of user information and characteristics, which are not always available due to privacy and optional user authentication [20]. To solve this problem, the process of providing recommendations should consider the user's recent interactions in the system and extract his/her behavior patterns.

The purpose of SBRS is to reduce the effect of the abovementioned problems. These types of systems try to ensure that information related to the structure of sessions and users' short-term preferences are not ignored. Session-based recommender system provide recommendations based on session data without the need to access long-term user data and preferences. Due to the availability of input data required for session-based recommender systems and the compatibility of the features of these types of systems with real-world problems, SBRS has gained the attention of many researchers. These types of systems have wide applications in various fields, such as Web pages, tourism, news, hotels, media, etc. Of course, it should be noted that there are very narrow boundaries between session-based and sequential recommender systems [22]. Session-based recommender systems are modeled based on session data, and sequential recommender systems are modeled on sequential data. Further discussion of the differences between these systems is provided in Sect. 1.4.

The goal of session-based recommender systems is to predict one or a group of items of a session or to predict the next session based on learning the dependencies within each session or between several sessions. These dependencies are recognized based on co-occurrences of interactions in a session [20]. In contrast, in sequential recommender systems, items are predicted based on learning sequential dependencies between consecutive items in different sessions [23]. Figure 1.2 presents the general process of a session-based recommender system, which shows the functionality of this type of system in the purchasing activities of different users.

From another perspective, the main difference between session-based recommender system and traditional recommender systems can be summarized in the three factors of data, task, and user, as shown in Table 1.2.

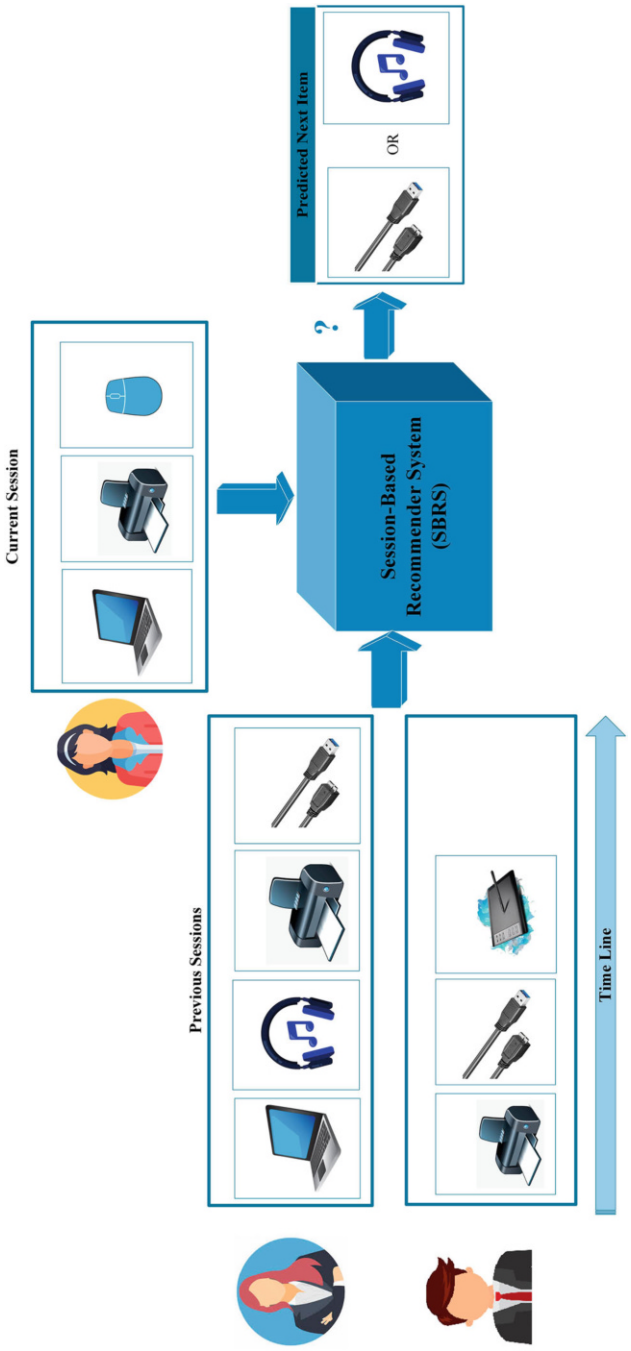

Fig. 1.2 SBRS process in the purchasing activities of different users

Table 1.2 Differences between session-based recommender systems and traditional recommender systems

Major factors	Traditional recommender system	Session-based recommender system
Data	– User-item rating matrix – User profile – Item features	Timed and organized sequence of action/interaction in sessions
Task	Making time-independent recommendations for users' long-term preferences	Make recommendations for the current session that fit the user's short-term interactions
User	Known user profile and usually available	The profile is usually anonymous

1.3.1 Basic Concepts of SBRS

Session-based recommender systems consist of entities, including user, item, and user-item interaction. In this subsection, user/item, action/interaction, and session concepts are briefly explained as follows:

- **User/item**: A user is a person who takes actions in a system related to items, for example, clicks, makes purchases, and receives the results of recommendations. Each user has a unique identifier, and a set of explicit or implicit attributes is considered to describe her/him. Of course, the users' session information may not always be available for two reasons: (1) it is not recorded due to privacy protection, and (2) some users may not log in through the authentication system and may be anonymous. An item is also an entity in a system that needs to be recommended, like a product. Each item is treated with a unique identifier and a set of features to provide item description information.
- **Action/interaction**: Action is often performed by a user on an item in a session, for example, clicking on an item. Each action is provided with a unique identifier and a set of attributes to provide information about it. Action has several different types: click, view, buy, etc. Interaction is the most basic unit in the session. An interaction is a ternary-tuple formed based on the *action* performed by a *user* on a specific *item*. If the user's information is not available, the interaction is anonymous.
- **Session**: In session-based recommender systems, the session is the basic unit of data organization for data analysis and providing recommendations. The meaning of the word "session" in the Oxford Dictionary means "a period of time that is spent doing a particular activity," but in [20], the specialized meaning of the word "session" in the field of recommendation systems is "a session is composed of multiple user-item interactions that happen together in a continuous period of time." Also, a session can be a set of events and activities that happen in a specific time period, for example, a set of purchased items or a group of music listened.

There are different formalizations for session-based recommender systems. Here, we present a formalization at an abstract level generally suitable for these systems. Suppose $l = \{int_1, \ldots, int_n\}$ is a list of n interactions; each of these interactions

consists of an item and the corresponding action. Considering the systems that are built on single-type action sessions, each interaction in the set l is reduced to one item, and therefore, the interaction set l becomes the item set $l_i = \{i_1, \ldots., i_n\}$ $(i_j \in I)$, where I is the set of items. L is also a set of possible interaction lists derived from the set of candidate *items* I and the set of actions A.

Now consider that U is a set of users. Unlike traditional recommender systems, our goal is not to predict a utility score for each $i \in I$ and for each $u \in U$, but to calculate an ordered list of the set L for each user, where each element $l \in L$ corresponds to $i \in I$. For this purpose, we define a utility function f to compute the score of a certain sequence l for user u based on Eq. (1.1):

$$l_u = \arg\ \max\ f(u, l), \quad u \in U, l \in L \tag{1.1}$$

According to the above Eq. (1.1), the goal of an SBRS is to select the recommended interaction list $l_u \in L$ to maximize the utility score for the user u. The utility function is applied to the list of interactions to optimize the list of candidates as a whole and not as a single interaction (item). In general, f function is not limited to specify the utility score for individual items but for entire ordered lists of items. This makes it possible to consider other aspects of usefulness in session-based recommender system problems, including the diversity of the collection as a whole and the quality of the order of recommendations in terms of transitions between objects.

For each session, a set of attributes are considered, which are listed in Table 1.3:

Based on the type of recommendations, SBRS can be divided into three categories: next-item recommender systems, next-partial-session recommender systems, and next-session (next-basket) recommender systems. The goal of the next-item recommender systems is to suggest the next possible interaction in the current session by modeling the dependencies within the session. According to the known parts of a session, the next-partial-session recommender systems recommend the unknown parts. Indeed, based on intra-session dependencies, it recommends all remaining interactions to complete the current session, for example, predicting all subsequent items to complete a cart given the items purchased. Considering past sessions, the goal of the next-session recommender systems is based on modeling dependencies between sessions. Sometimes, dependencies between sessions are also included in the previous two types to improve the performance of the recommendation.

1.3.2 Challenges of SBRS

The two major challenges in session-based recommender systems are related to data and task modeling. Regarding data challenges, it should be considered that each dataset in session-based recommender systems has a hierarchical structure, which includes session level, item level, and item feature level, which are the main core of

Table 1.3 Attributes of a session in session-based recommender systems

Session attributes	Description
Session length	A session is determined based on the number of interactions. Different lengths of sessions in a system affect the efficiency of that system
Action type	Specifies the actions performed by users in a session. Some systems consider only one action, for example, clicking, while other systems record several different actions, such as clicking, purchasing, etc. These actions may be interdependent. Therefore, the number of action types in a session determines whether the dependencies within the session are homogeneous (based on a single action type) or heterogeneous (based on multiple action types)
Inner order	It refers to the order of interactions within the session. Usually, the interactions of different sessions can be unordered, be ordered, or have a flexible order
User information	It contains the user ID in the system or the attributes of that user. User information plays an essential role in connecting sessions to each other and accessing users' long-term interests, but usually, users are anonymous, and sessions do not contain user information
Inter-session contextual data	The set of recent sessions that occurred before the current session is the context data for the current session. This type of data shows dependencies and connections between sessions
Intra-session contextual data	Contextual data within each session that recommends unknown data includes items that are known in the same session. Intra-session contextual data specifies intra-session dependencies

learning models in this system. The challenges related to different levels of data can be divided into the following four categories: heterogeneity within each level, coupling within each level, complexity within each level, and interactions between different levels. Each item can be introduced by several heterogeneous features such as price, country of manufacture, etc., and each item feature can have different values, some of which may be repeated more than others. It is a set of items and user interactions that form a session.

The task modeling challenge begins with extracting information from the session, item, and item features and continues until the development of an intelligent model suitable for the task requirements and its evaluation approach.

The challenges related to the data are:

- **Heterogeneity within each level**: Different elements in each level have different specifications and features. Therefore, they should be examined from different approaches, and each should be modeled based on appropriate methods. At the item feature value level, the distribution of values is different, and one value may be more iterated than the other. At the item features, there are usually different features that are heterogeneous and cannot be modeled with the same method. For example, the manufacturing country of an item and its price are different from each other. At the item level, the items related to the same sessions have different distributions, and some of them may be known and frequent, but other items are rarely selected. Heterogeneity at the session level means that the connection of

each session with the current session of the user is different because the sessions have different contexts. Some of them may even be irrelevant. Finally, the various contextual factors that influence the evolution of sessions may include time, location, season, etc. Since these factors are heterogeneous, they cannot be modeled similarly.

- **Coupling within each level**: Different levels of data in session-based recommender system depend on the interactions between their elements. At the level of feature values, interactions between different feature values of each item may cause coupling in case the values belong to one feature or are related to different features of an item [24]. For example, the type of product affects its price. Coupling at the feature level of an item means that one feature may affect another, or a combination of several features may lead to the extraction of a specific pattern. Common interactions between items in a session led to the coupling between them. For example, some items are purchased together in a store. The interactions between different sessions also affect each other, and the transactions of each session are effective on the transactions of the next session. For example, if a customer buys a car in one session, she/he will probably have a transaction related to car insurance in the next session. The coupling between different domains is also made based on the interaction between them; for example, after users watch a movie, they are likely to search for the music of that movie. Contextual couplings also bring about the effect of various contextual factors on user transactions. For example, the products purchased by a user in winter are different from those purchased in summer.
- **The other complexity within each level**: In addition to coupling and heterogeneity, other challenges lead to data complexity in session-based recommender system. For example, the complexities inherent in an item level may include implicit dependence, lack of coordination, or lack of balance of items at the session level, and there are complexities including long-term dependencies related to previous sessions and session modeling under one or more specific contextual factors such as time and location [25].
- **Interactions between different levels**: The features of an item have mutual effects on the occurrence of that item in a transaction. For example, items that belong to the same category are more likely to be selected together. Also, the mutual effects of data on two levels of the session and item are such that previous sessions have an effect on the item selected in the current session. For example, a user who purchased a house in the previous sessions may choose items related to home appliances in the next session.

The challenges related to the task modeling are:

- **Extraction of relevant information**: the effective extraction of information about different levels of session, item, and item features can be achieved using different deep learning methods or transformers. Some of these patterns are complicated, such as spatiotemporal patterns, which makes it difficult to choose or make an appropriate algorithm for this task. In many cases, items have rich features such as images and text descriptions that can be used to model sessions.

The appropriate method of how to use these features in related models is an important challenge. Considering the semantic-level structural information among the items can also lead to the discovery of additional external knowledge sources that will be effective in the performance of SBRS.

- **Modeling user preferences**: this modeling extends far beyond a consecutive time pattern in the transition of item choices. Recent research on session-based recommender systems has mostly focused on capturing sequential patterns using the attention mechanism, which is efficient for the session's natural sequence sorted by time, but has various problems with the complicated item transition pattern. On the other hand, users' preferences change dynamically, which can make the modeling of these preferences difficult in final SBRS.
- **Considering cross-sessions**: instead of extracting sequential patterns in individual sessions, information modeling in cross-sessions can result in more complex dependency relations between items. However, it is not easy to use cross-session information due to the anonymity of session data, specially in cases where relations between different sessions are prevented. Basically, the use of shared information in neighboring sessions that has already been created by other users and reflects the user's intentions similar to the current session can help the performance of SBRS in the current session.
- **Evaluation process**: identifying standard evaluation protocols and the existence of widely used baselines to compare the performance of the proposed methods is another challenging factor. The simulated or real datasets extracted from users' actual behavior in different sessions are another one.

1.3.3 Session-Based vs. Sequential vs. Session-Aware Recommender Systems

User preferences change and evolve. Given this fact, it is better to incorporate time into the recommendation process and recognize patterns in sequential user behavior more effectively than static recommendations. The research proposed in this field typically distinguish between a sequence, which is a time-ordered list of interactions without defined time intervals, and a session, which is a list of ordered or non-ordered interactions with a defined boundary and usually covering a relatively short time interval. To clarify the boundaries of sequential, session-based, and session-aware recommender systems, we briefly explain them in the following:

- **Sequential or sequence-aware recommender system (SRS)** employs data with a specific sequence and order and are not necessarily based on sessions. For example, data with a timestamp or sorted by time or date is considered. These types of systems attempt to explicitly recognize sequential dependencies, such as behavioral patterns, and discover information within interactions that can be realized by considering interactions as sequences of events. Different types of patterns, including sequential, co-occurrence, and distance patterns, can be

considered [26]. Sequential patterns associate interactions in a specific order, while co-occurrence patterns only care if two interactions happen together. Distance patterns are less conventional, and they try to identify necessary temporal distances before recommending items. On the other hand, these types of systems usually consider a long-term sequence of user interactions and provide recommendations based on these interactions [22]. These interactions usually specify the selected item and the type of user behavior, for example, clicking on item 1 or purchasing item 2. Therefore, in addition to users, items also have additional data, and time-ordered events are extracted based on the time-based user-item matrix [27]. The goal is to predict the user's next item based on all the user's preferences in his/her long-term profile [28].

- **Session-based recommender system (SBRS)** takes a list of user interactions as input, which are mostly grouped into anonymous sessions. For example, sessions can be related to a session of music listened to on a music streaming Web site or a shopping session on an e-commerce site [28]. In fact, these types of systems only focus on session data that include their short-term events [22]. Compared to other recommender systems, the main approach of this type of recommender system is that users are not followed among different sessions and the systems provide recommendations only based on a list of short-term users' preferences, including their recent interactions [27]. This feature is essential for Web sites that face the problem of new users without any interactions or users who have not been authenticated. Nowadays, researchers in the field of session-based recommender systems are very active, and their works are very relevant to real-world problems [28].
- **Session-aware recommender system (SARS)** is a special and personalized type of session-based recommender system [28]. In both types of systems, the grouping of user interactions is done in specific sessions, and the goal of both is to predict the user's favorite items. However, in session-aware recommender systems, the users are not anonymous, and their previous actions can be obtained through the events and interactions of the previous sessions, and based on them, the next interactions of the current session can be predicted. Therefore, in addition to users having IDs, sessions also have specific IDs. These types of systems usually use a combination of short-term and long-term preferences (more attention to short-term) of users to recommend new items [27].

Figure 1.3 shows the similarities and differences between SBRS, SRS, and SARS.

However, using the above terms in the recommender systems literature is not always consistent. Sometimes, the term session-based recommendation is also used for situations where long-term preferences are available, for example, in [29], which is a fundamental research in the domain of session-based recommender systems. Several authors also use the term sequential recommender systems in session-based recommendation scenarios. In fact, the problem of session-based recommendations can be considered from two sequential aspects, both in the sense that the goal is to predict the next interactions in a session and the available data are chronologically

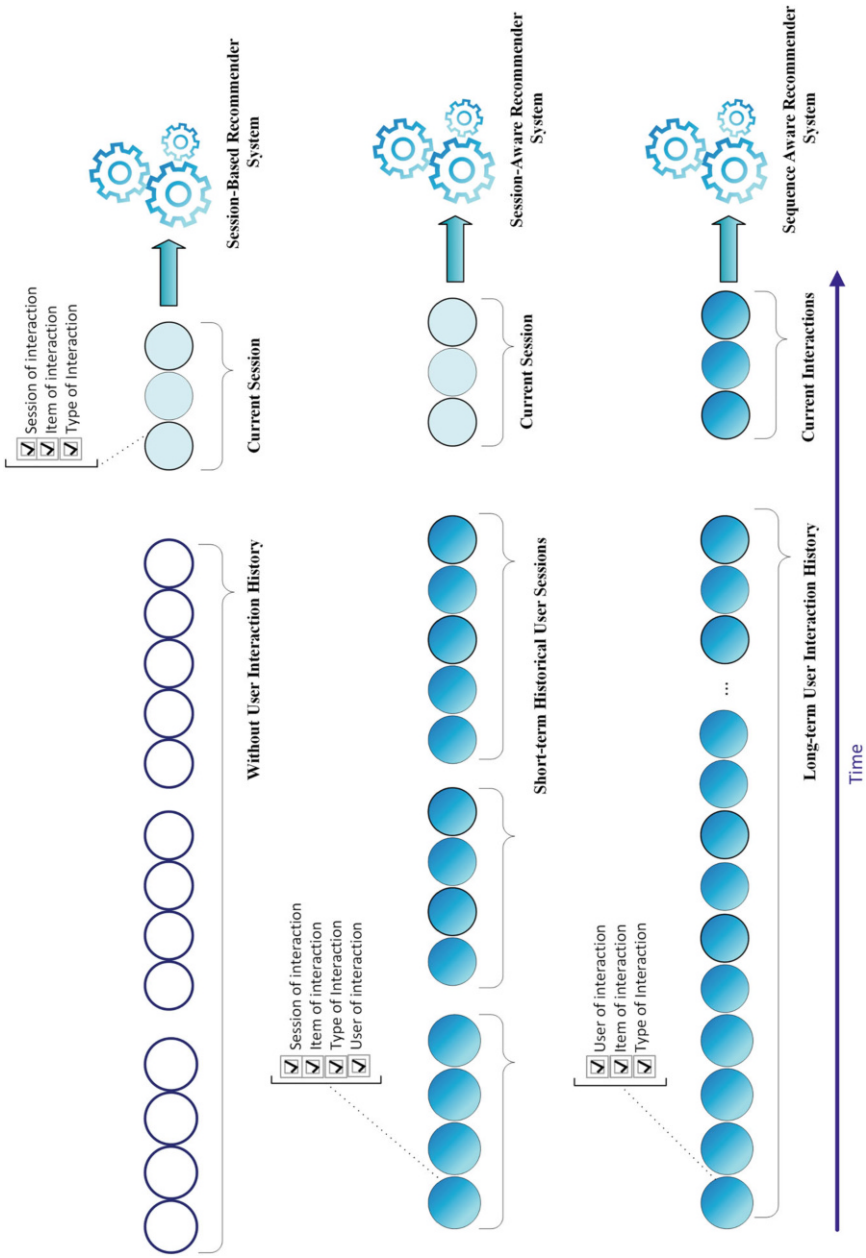

Fig. 1.3 SBRS vs. SRS vs. SARS

sequential. However, not all sequential approaches are necessarily session-based. Instead, it may be based on the user's preferred information and have a long-term timestamp.

1.4 Session-Based Recommender System Approaches

The goal of session-based recommender system is to predict unknown items of a session or to predict the user's next session based on modeling the complex relations within a session or between sessions. There are two different perspectives to study these types of systems. In the first view, the user's next items are recommended based on the data of the user's current session, called intra-session context data. In the second view, the next session or a part of the next session of a user is predicted based on the inter-session context [20, 30].

On the other hand, SBRS approaches are divided into two general categories in terms of the techniques used: traditional SBRS approaches and deep learning SBRS approaches. Each of these categories is divided into several more detailed subcategories. For example, traditional SBRS approaches include pattern/rule mining, K-nearest neighbors (KNN), Markov chain, generative probabilistic model, and latent representation approaches. Deep learning SBRS approaches are mainly divided into two subcategories: basic and hybrid/advanced deep neural networks. Basic deep neural network methods include discriminative and generative models, while hybrid/advanced deep neural network approaches include graph-based methods, attention-based models, deep reinforcement learning, and hybrid.

Figure 1.4 shows the taxonomy of the session-based recommender system approaches. Each of these models is briefly explained in the subsequent subsections.

1.4.1 Traditional SBRS

These approaches are based on data mining or machine learning techniques to recognize the latent dependencies in sessions for the recommendation process. The main idea of this type of approach is to detect sequences of data related to user sessions by using recognition and mining of patterns and dependencies between them to provide recommendations to users. These approaches are divided into five separate categories, which are discussed in the following subsections:

1.4.1.1 Pattern/Rule Mining

In general, there are two types of pattern/rule mining-based approaches for session-based recommender systems: (1) frequent pattern/association rule mining approaches and (2) sequential pattern mining approaches. These approaches only

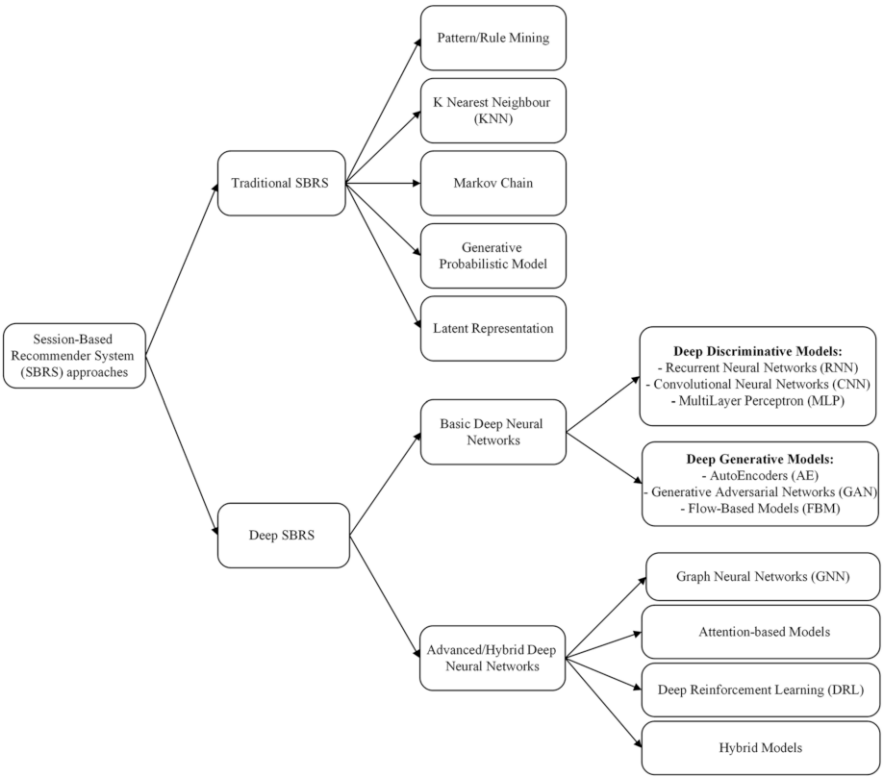

Fig. 1.4 Taxonomy of session-based recommender system approaches

apply to sessions whose data is based on a single type of action and all dataset actions are the same.

- **Frequent pattern/association rule mining-based approaches:** In these systems, first, frequent patterns or association rules are mined; then, user sessions with the detected patterns and rules are identified; and finally, next-item recommendations are delivered based on the results. Most users are assumed to act based on commonly applied and frequent patterns. For example, the review of users' purchases shows that customers usually buy mobile and handsfree together, so based on the model of mobile, handsfree can also be recommended to customers who buy mobiles. These types of systems can be applied to data without order and sequence.
- **Sequential pattern mining-based approaches:** Recommender systems based on sequential patterns are suggested for processing ordered data, and their items are based on temporal factors. Such systems first detect sequential patterns from the dataset, and then based on the sequential pattern mining and the order of items selected by the user, they recommend the next item. Recommender systems based on sequential patterns have two fundamental differences from pattern-based

recommender systems. The first difference is that recommender systems based on sequential patterns usually make cross-session recommendations by utilizing inter-session dependencies, while in most cases, pattern-based recommender systems utilize intra-session dependencies to make inner-session recommendations. The second difference is caused by the order of the data, because recommender systems based on sequential patterns consider the orders over sessions, which are appropriate for sequential data. Techniques such as previous weighted sequences of users or combining with collaborative filtering have also been proposed to improve approaches using basic sequential pattern recognition methods.

1.4.1.2 K-Nearest Neighbors

K-nearest neighbors (KNN) is a supervised machine learning algorithm applied to solve various problems such as classification and regression. A session-based recommender system using the KNN method considers each interaction as an item. In this type of approach, first, interactions or sessions that are similar to the current interaction or session are captured. Then, a score is calculated for each candidate interaction based on how similar and relevant it is to the current interaction and based on whether similarity is considered between items or between sessions.

KNN-based approaches are divided into two categories: item-KNN or session-KNN. For both approaches, the set of neighboring items and sessions can usually be computed in advance to speed up prediction time [31]:

- **Item-KNN approaches**: recommend the K items that are most similar to the current item, corresponding to the co-occurrence of items in other sessions.
- **Session-KNN approaches**: first, calculate the similarity between the current session and other sessions to detect a set of K neighboring sessions. Then, the score of each candidate item is determined based on that. These approaches consider all the context of the session and obtain more accurate information for the recommendation process than the item-KNN that only considers the current item of the current session.

1.4.1.3 Markov Chain

A Markov chain or Markov process is a stochastic model that describes a sequence of possible events, where the probability of each event depends only on the state obtained in the previous event. SBRS using Markov chains first model the transitions between the interactions of one or more sessions using Markov chains and then predict the next possible interactions or the next session. Unlike recommender systems based on sequential patterns, which remove items that are less frequent to provide recommendations, in recommender systems based on Markov chains, all items are considered. Most approaches in this field use the first-order Markov chain

to reduce the complexity of the model. According to how the transition probabilities are calculated based on explicit observations or latent space, Markov chain-based approaches can be divided into basic Markov chain and latent Markov embedding approaches:

- **Basic Markov chain approaches**: The process of basic Markov chain approaches consists of four main steps: The first step includes computing the first-order transitional probability over interactions in the training data, the second stage predicts the transition paths between interactions, the third stage matches the context of the session with the predicted paths, and the fourth stage provides recommendations based on the results of the previous stages. In fact, items with higher probability are added to the list of recommendations. To improve the results, some studies have employed techniques such as combining the first-order and second-order Markov models, creating a hidden Markov model based on probabilistic models, and factorization of the transition probability matrix.
- **Latent Markov embedding approaches**: Unlike the basic Markov chain method, which computes the transition probabilities based on explicit observations, the approaches based on the latent Markov embedding method first embed the Markov chain in a Euclidean space and then compute the transition probability between items based on Euclidean distance. This method can include unobserved transitions, which significantly reduces the problem of data sparsity in cases with limited observed data.

1.4.1.4 Generative Probabilistic Model

Generative probabilistic model approaches typically first infer latent categorization of items in sessions and then learn transitions between these latent categories within or between sessions. Then, the next latent categorization is predicted using the learned transitions. Finally, they predict specific items as the next item, conditional on the predicted latent ranking of the items. Latent topic models are commonly used to infer latent categories and transitions between them.

1.4.1.5 Latent Representation

In session-based recommender systems, latent representation approaches using shallow models generate a latent representation with low dimensions. This representation is created for each interaction in a session. The learned informative representations encode the dependencies between these interactions and are used for subsequent session-based recommendations. One of the most popular models used in this type of approach is the latent factor model. SBRS based on latent factor models first determines a factorization model to decompose the observed transition matrix of interactions into their latent representations and then uses the resulting latent representations to estimate unobserved transitions for the next session-based

recommendations. Interactions in these types of approaches are considered items. One of the widely used methods in this area is matrix factorization. Factorization-based approaches first factorize the item-user matrix or the item-item co-occurrence matrix into the latent representation vector of each item and predict the next items using the latent representations. These approaches usually utilize factorization machines such as matrix factorization in recommender systems based on collaborative filtering to factorize the matrix of user-item interactions into latent factors of users and items.

1.4.2 Deep Learning SBRS

In this section, we aim to provide a quick look at different deep learning models that have been used in SBRS. Hence, the readers gain a preliminary perspective on the main topic of this book, which will be discussed in the following chapters.

Approaches using deep learning employ various neural network models to learn complicated relations between items in each session or between different sessions. These types of approaches are divided into two groups based on the number of layers and the depth of the model: approaches based on shallow neural networks and approaches based on deep neural networks.

Shallow neural networks are a network architecture with a limited number of layers and so a limited modeling power. Deep neural networks are used to learn an optimized combination of different representations to predict and recommend the next item or session. Deep learning provides two specific goals in the domain of recommender systems: processing features of items and users and modeling relations and interactions between users and items [32]. On the other hand, it provides several benefits in the scope of session-based recommender systems, including the ability to create non-linear models, extracting and engineering features automatically for different types of data, high capability of sequential modeling for sequential data, high scalability, and flexibility for modeling hybrid recommender systems [33]. In SBRS, various deep neural network techniques are used to detect dependencies between the interactions of a session and complex and extensive relations between different sessions. Therefore, the motivation of researchers to provide SBRS based on deep learning techniques has increased in recent years.

Deep learning approaches in various fields of SBRS are performed to achieve these specific goals: using deep neural networks to process the features of items and users and modeling the interactions between users and items. The strengths that have led session-based recommender systems to deep learning approaches are as follows:

- The ability to generate non-linear models
- No need for engineering and diagnosing features manually for data like text, image, and sound
- The significant capability of sequential modeling for sequential data

– The high scalability and high flexibility for modeling hybrid session-based recommender systems
– Acceptable accuracy in obtaining results
– The ability to learn unlabeled data

Session-based recommender systems using deep learning approaches can be divided into two categories: basic deep neural networks and hybrid/advanced deep neural networks:

- **Basic deep neural network approaches** utilize deep discriminative models such as recurrent neural networks (RNN), convolutional neural networks (CNN), multilayer perceptron (MLP), and deep generative models, such as autoencoders (AE), generative adversarial networks (GAN), and flow-based models (FBM). The first use of deep neural networks in SBRS was in 2015 by Hidasi et al., when a model based on RNN was presented [34]. Many researchers in the field of deep learning session-based recommender systems use RNN and its variants, including GRU and LSTM models. In these session-based recommender systems, user clicks or interactions are given as input to the system and are transformed into meaningful data structures using an embedding method. Then, a recurrent neural network is used to model the data and detect their dependency relations. Finally, the fully connected layer is used before the output layer to make the model more stable. Due to their sequential nature, recurrent neural networks have a high potential to analyze the sequential dependencies between data in user sessions. The capability to model dynamic behaviors over time in SBRS has made recurrent neural networks a desirable solution in this field. Most of the research uses GRU type of RNNs because the number of gates and parameters of the LSTM model is larger and it has higher computational complexity. RNN and its variant models in SBRS are discussed in detail in Chap. 3.

  Some approaches related to SBRS use the convolutional neural network model. Using CNN is suitable for user session data in two ways: (1) The order of items in one session or between different sessions of users can be easily implemented and modeled on convolutional neural networks. (2) Convolutional neural networks have a high capacity to learn local features of an area or special relations between different areas, based on which they can recognize dependencies that other models usually ignore. In such systems, to learn and model data related to users and items, the inputs should be properly embedded so that temporal and spatial patterns between them can be correctly identified by successively using convolution and pooling layers in CNNs. The user's favorite items are predicted based on the features obtained from the input data and the dependencies between them. CNN models in SBRS are discussed in detail in Chap. 3.

  MLP-based approaches are usually employed to learn the optimal combination of different representations to create a complex representation of the session context for later recommendations. Unlike RNN-based approaches, MLP-based approaches are mainly suitable for unordered session data due to their inability to model sequential data. It is necessary to mention that MLPs are not mainly used alone in SBRS and are usually employed as complementary modules.

Approaches based on deep generative models for SBRS provide recommendations by generating the next interaction or session through a carefully designed generative strategy. Deep generative models pursue two goals: learning practical and correct representation of data using unsupervised methods and learning probabilistic distributions of data and related classes. These approaches also are classified as autoencoders (AE), generative adversarial networks (GAN), and flow-based models (FBM). Generative models in SBRS are discussed in detail in Chap. 4.

- **Hybrid/advanced deep neural network approaches** Over time, to reduce challenges such as complex dependencies of variables in different time steps, cold start problem, data sparsity, and also more effective optimization of complicated session-based recommender systems, more advanced deep learning approaches were presented to improve the recommendation process. These approaches include advanced deep neural networks such as graph neural networks (GNNs), attention-based models, deep reinforcement learning, and hybrid models. These models are discussed in detail in Chap. 5.

  Although conventional deep learning techniques have been successful in various fields, most of their data are in Euclidean space, while many data are inherently better represented by graph structures [35, 36]. In SBRS, it is possible to model the sequential behaviors and interactions of users with a graph and learn the relations between them using deep graph neural network models. In these approaches, given a dataset containing several sessions, each session is mapped to a chain on the graph. Each interaction in a session acts as a node in the corresponding chain, where an edge is created to connect each pair of adjacent interactions in the session. The constructed graph is then fed as input to the GNN to learn rich information embedding for each node by encoding complex transitions on the graph into embeddings. Finally, these learned embeddings are fed into the prediction module for session-based recommendations. Graph neural networks can also be combined with CNN and RNN models. Different types of graph neural network-based approaches can be further divided into GCN or GAT. GNN and its variant models in SBRS are discussed in detail in Chap. 5.

  Attention-based session recommender systems provide an attention mechanism for discriminative element exploitation in a session context. For accurate recommendations, these systems try to generate an informative representation from the context of the session. By integrating the attention mechanism, a session-based recommender system can emphasize items that are most relevant to the next interaction or session and reduce the interference of irrelevant items in the session context. In general, an attention model mainly consists of two steps: calculating attention weights for the relevant weights of interactions and aggregation, which aggregates the embeddings of all interactions of the session to learn their weights. It is worth mentioning that attention mechanisms are not mainly used alone in SBRS and are usually utilized together with basic models such as GNN [37, 38], CNN/LSTM [39], CNN/GRU [40], and MLP [41].

  The deep reinforcement learning (DRL) approach is focused on goal-directed learning through interaction, where the learning agent, through trial and error and

receiving rewards and punishments, determines which action receives the most rewards. In session-based recommender systems that dynamically recommend items to users, deep reinforcement learning methods are used to maximize expected long-term cumulative rewards. Such approaches can optimize recommendations for long-term user interactions instead of maintaining a short-term goal of optimizing the process of providing immediate recommendations to the user. Deep reinforcement learning methods enable recommender agents to learn optimal recommendation policies to recommend items to users [42]. An SBRS that uses reinforcement learning aims to learn optimal recommendation strategies through trial and error and receive reinforcement data for recommended items from user feedback [43]. In this way, these types of systems can continuously update their strategies during interaction with users until they reach the best state that meets their dynamic preferences. DRL models in SBRS are discussed in detail in Chap. 5.

Finally, hybrid approaches mainly include several primary deep neural network models to take advantage of each to model the various complex dependencies embedded in the session data. Each of the basic models recognizes one or more types of dependencies. In fact, a hybrid session-based recommender system performs two main steps: (1) learns different types of dependencies using different base models and (2) attentively integrates the learned dependencies to provide more accurate recommendations. These models are also discussed in detail in Chap. 5.

1.5 Conclusion

Session-based recommender systems provide recommendations based on session data without the need to access long-term user data and transactions. Due to the availability of input data required for SBRS and the compatibility of the features of these types of systems with real-world problems, SBRS has attracted the attention of many researchers.

In this chapter, we briefly reviewed the preliminary concepts of recommender systems and session-based recommender systems and presented the significant differences between these two systems in detail. Then, the fundamental components and important attributes of session-based recommender systems, along with their specific challenges, were discussed.

When time and sequence of events are incorporated into the recommendation process, there will be three major approaches, session-based, sequential, and session-aware recommender systems, which were fully elaborated on this chapter.

Since different research has been proposed in the scope of SBRS, each of these models was briefly explained to provide a quick look at different deep learning models used in session-based recommenders.

References

1. Dietmar Jannach, Massimo Quadrana, and Paolo Cremonesi. "Session-based recommender systems." In Recommender Systems Handbook, pp. 301-334. Springer, New York, NY, 2022. https://doi.org/10.1007/978-1-0716-2197-4_8
2. Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. "Recommender systems survey." Knowledge-based systems 46 (2013): 109-132. https://doi.org/10.1016/j.knosys.2013.03.012
3. Qian Zhang, Jie Lu, and Yaochu Jin. "Artificial intelligence in recommender systems." Complex & Intelligent Systems 7 (2021): 439-457. https://doi.org/10.1007/s40747-020-00212-w
4. Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. "Recommender system application developments: a survey." Decision support systems 74 (2015): 12-32. https://doi.org/10.1016/j.dss.2015.03.008
5. Eva Zangerle, and Christine Bauer. "Evaluating Recommender Systems: Survey and Framework." ACM Computing Surveys 55, no. 8 (2022): 1-38. https://doi.org/10.1145/3556536
6. Matteo Marcuzzo, Alessandro Zangari, Andrea Albarelli, and Andrea Gasparetto. "Recommendation Systems: An Insight Into Current Development and Future Research Challenges." IEEE Access 10 (2022): 86578-86623. https://doi.org/10.1109/ACCESS.2022.3194536
7. Fatemeh Alyari, and Nima Jafari Navimipour. "Recommender systems: a systematic review of the state of the art literature and suggestions for future research." Kybernetes (2018). https://doi.org/10.1108/K-06-2017-0196
8. Yehuda Koren, Steffen Rendle, and Robert Bell. "Advances in collaborative filtering." Recommender systems handbook (2022): 91-142. https://doi.org/10.1007/978-1-0716-2197-4_3
9. Wang Juan, Lan Yue-xin, and Wu Chun-ying. “"Survey of recommendation based on collaborative filtering”" In Journal of Physics: Conference Series, vol. 1314, no. 1, p. 012078. IOP Publishing, 2019. https://doi.org/10.1088/1742-6596/1314/1/012078
10. Pasquale Lops, Dietmar Jannach, Cataldo Musto, Toine Bogers, and Marijn Koolen. “Trends in content-based recommendation: Preface to the special issue on Recommender systems based on rich item descriptions” User Modeling and User-Adapted Interaction 29 (2019): 239-249. https://doi.org/10.1007/s11257-019-09231-w
11. Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. “"A survey on knowledge graph-based recommender systems”" IEEE Transactions on Knowledge and Data Engineering 34, no. 8 (2020): 3549-3568. https://doi.org/10.1109/IAEAC50856.2021.9390863
12. Petra Perner. "Case-based reasoning–methods, techniques, and applications." In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 24th Iberoamerican Congress, CIARP 2019, Havana, Cuba, October 28-31, 2019, Proceedings 24, pp. 16-30. Springer International Publishing, 2019. https://doi.org/10.1007/978-3-030-33904-3_2
13. Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. "Constraint-based recommender systems." Recommender systems handbook (2015): 161-190. https://doi.org/10.1007/978-1-4899-7637-6_5
14. Saurabh Kulkarni, and Sunil F. Rodd. “"Context Aware Recommendation Systems: A review of the state of the art techniques”" Computer Science Review 37 (2020): 100255. https://doi.org/10.1016/j.cosrev.2020.100255
15. Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. "A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation." IEEE Transactions on Knowledge and Data Engineering (2022). https://doi.org/10.1109/TKDE.2022.3145690
16. Ido Guy. "Social recommender systems." In Recommender Systems Handbook, Second Edition, pp. 511-543. Springer US, 2015. https://doi.org/10.1007/978-1-4899-7637-6_15
17. Hossein Tahmasebi, Reza Ravanmehr, and Rezvan Mohamadrezaei. "Social movie recommender system based on deep autoencoder network using Twitter data." Neural Computing and Applications 33 (2021): 1607-1623. https://doi.org/10.1007/s00521-020-05085-1

18. Hirad Daneshvar, and Reza Ravanmehr. "A social hybrid recommendation system using LSTM and CNN." Concurrency and Computation: Practice and Experience 34, no. 18 (2022): e7015. https://doi.org/10.1002/cpe.7015
19. Francesco Ricci, Rokach, Lior, Shapira, Bracha and Kantor, Paul B.. Recommender systems handbook. New York; London: Springer, 2022, ISSN: 978-1-0716-2197-4, https://doi.org/10.1007/978-1-0716-2197-4
20. Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet A. Orgun, and Defu Lian. "A survey on session-based recommender systems." ACM Computing Surveys (CSUR) 54, no. 7 (2021): 1-38. https://doi.org/10.1145/3465401
21. Gabriel De Souza P. Moreira, Dietmar Jannach, and Adilson Marques Da Cunha. "Contextual hybrid session-based news recommendation with recurrent neural networks." IEEE Access 7 (2019): 169185-169203. https://doi.org/10.1109/ACCESS.2019.2954957
22. Dietmar Jannach, Bamshad Mobasher, and Shlomo Berkovsky. "Research directions in session-based and sequential recommendation: A preface to the special issue." User Modeling and User-Adapted Interaction 30 (2020): 609-616. https://doi.org/10.1007/s11257-020-09274-4
23. Ali Noorian, Ali Harounabadi, and Reza Ravanmehr. "A novel Sequence-Aware personalized recommendation system based on multidimensional information." Expert Systems with Applications 202 (2022): 117079. https://doi.org/10.1016/j.eswa.2022.117079
24. Longbing Cao. "Coupling learning of complex interactions." Information Processing & Management 51, no. 2 (2015): 167-186. https://doi.org/10.1016/j.ipm.2014.08.007
25. Longbing Cao. "Data science: challenges and directions." Communications of the ACM 60, no. 8 (2017): 59-68. https://doi.org/10.1145/3015456
26. Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. "Sequence-aware recommender systems." ACM Computing Surveys (CSUR) 51, no. 4 (2018): 1-36. https://doi.org/10.1145/3190616
27. Malte Ludewig. "Advances in session-based and session-aware recommendation." PhD diss., Dissertation, Dortmund, Technische Universität, 2020, 2020.
28. Sara Latifi, Noemi Mauro, and Dietmar Jannach. "Session-aware recommendation: A surprising quest for the state-of-the-art." Information Sciences 573 (2021): 291-315. https://doi.org/10.1016/j.ins.2021.05.048
29. Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. "Personalizing session-based recommendations with hierarchical recurrent neural networks." In proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 130-137. 2017. https://doi.org/10.1145/3109859.3109896
30. Lemei Zhang, Peng Liu, and Jon Atle Gulla. "A deep joint network for session-based news recommendations with contextual augmentation." In Proceedings of the 29th on Hypertext and Social Media, pp. 201-209. 2018. https://doi.org/10.1145/3209542.3209557
31. Dietmar Jannach, and Malte Ludewig. "When recurrent neural networks meet the neighborhood for session-based recommendation." In Proceedings of the eleventh ACM conference on recommender systems, pp. 306-310. 2017. https://doi.org/10.1145/3109859.3109872
32. Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. "Deep learning based recommender system: A survey and new perspectives." ACM computing surveys (CSUR) 52, no. 1 (2019): 1-38. https://doi.org/10.1145/3285029
33. Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. "DKN: Deep knowledge-aware network for news recommendation." In Proceedings of the 2018 world wide web conference, pp. 1835-1844. 2018. https://doi.org/10.1145/3178876.3186175
34. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In Proceedings International Conference on Learning Representations, ICLR '16, 2016. https://arxiv.org/abs/1511.06939
35. Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. Graph neural networks. Springer Singapore, 2022. https://doi.org/10.1007/978-981-16-6054-2_3
36. Yao Ma, and Jiliang Tang. *Deep learning on graphs*. Cambridge University Press, 2021. https://doi.org/10.1017/9781108924184

37. Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. "Session-based recommendation with graph neural networks." In Proceedings of the AAAI conference on artificial intelligence, vol. 33, no. 01, pp. 346-353. 2019. https://doi.org/10.1609/aaai.v33i01.3301346
38. Zhiqiang Pan, Wanyu Chen, and Honghui Chen. "Dynamic graph learning for session-based recommendation." Mathematics 9, no. 12 (2021): 1420. https://doi.org/10.3390/math9121420
39. Qiannan Zhu, Xiaofei Zhou, Zeliang Song, Jianlong Tan, and Li Guo. "Dan: Deep attention neural network for news recommendation." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, pp. 5973-5980. 2019. https://doi.org/10.1609/aaai.v33i01.33015973
40. Jinjin Zhang, Chenhui Ma, Xiaodong Mu, Peng Zhao, Chengliang Zhong, and A. Ruhan. "Recurrent convolutional neural network for session-based recommendation." Neurocomputing 437 (2021): 157-167. https://doi.org/10.1016/j.neucom.2021.01.041
41. Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. "STAMP: short-term attention/memory priority model for session-based recommendation." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 1831-1839. 2018. https://doi.org/10.1145/3219819.3219950
42. Yuanguo Lin, Yong Liu, Fan Lin, Lixin Zou, Pengcheng Wu, Wenhua Zeng, Huanhuan Chen, and Chunyan Miao. "A survey on reinforcement learning for recommender systems." IEEE Transactions on Neural Networks and Learning Systems (2023). https://doi.org/10.1109/TNNLS.2023.3280161
43. Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. "Deep Reinforcement Learning for List-wise Recommendations." In 1st Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD 2019). 2019. https://arxiv.org/abs/1801.00209

Chapter 2
Deep Learning Overview

Abstract Among the various machine learning algorithms, deep learning has recently been dramatically used in different scopes. Deep learning models have been significantly employed in effectively extracting hidden patterns from vast amounts of data and modeling interdependent variables to solve complex problems. Since this book aims to discuss the session-based recommender system approaches using deep learning models, brief explanations of various deep neural networks are provided in this chapter. For this purpose, the history, basic concepts, advantages/applications, and fundamental models of deep learning are discussed.

Keywords Deep learning · Machine learning · Deep discriminative models · Deep generative models · Graph-based models

2.1 Introduction

The ability of the human brain to learn and solve different problems is impressive compared to the most powerful computers. For this reason, to increase the power of computers to solve more complex problems, the functioning of the human brain has been modeled to process different types of problem data. Indeed, an amazing evolution occurred in technology systems so that passive and static systems became active and dynamic and improved over time. This phenomenon was called machine learning, which allowed computers to learn.

Machine learning has been used in various research and utilized in different applications, such as text mining, spam detection, recommender systems, image classification, multimedia information retrieval, etc. Among the various machine learning algorithms, deep learning has recently been dramatically used in these applications. Deep learning employs neural networks and works based on the structure and function of neurons in the human brain system.

Today, deep learning models significantly affect extracting information or hidden patterns from massive data due to their higher capacity. Moreover, deep learning can solve complex problems and model interdependent variables compared to conventional machine learning approaches. Nowadays, deep learning technology is considered a hot topic in machine learning, artificial intelligence, and data science due to

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2_2

its ability to learn from different data types. Many companies, including Google, Microsoft, Apple, Meta, etc., are actively studying deep learning because it can provide significant results in various problems of analyzing large structured/unstructured datasets [1].

This chapter presents an overview of the basic concepts of deep learning, including its definition, history, advantages, and applications, and a comparison between the characteristics of deep learning and machine learning in Sect. 2.2. Then, a taxonomy of deep learning methods is presented, which includes the fundamental models of deep learning. Based on this classification, each of these models will be discussed and analyzed in the following subsections. In Sect. 2.3, deep discriminative models, including MLP, CNN, and RNN (GRU-LSTM), are described, and in Sect. 2.4, deep generative models such as AE, GAN, and methods based on the Boltzmann machine are explained. The fifth section describes graph-based models such as GNN and GCN.

2.2 Fundamentals of Deep Learning

2.2.1 History of Deep Learning

Although deep learning has become very popular in recent years, it has a long evolution process. Currently, mainstream deep learning approaches are based on neural networks, which have been researched for decades with varying levels of success. With the increase in hardware power and the emergence of big data, which provides much data for network training, it is possible to train networks with more than several hidden layers. Neural networks that consist of several layers are called deep networks. Currently, deep learning techniques are used in many domains, and the evolution of artificial intelligence and big data processing depends on deep learning methods.

The emergence of neural networks began in the early 1943s when Warren McCulloch and Walter Pitts developed a computer model focusing on the human neural system. They used a combination of algorithms and mathematics called "threshold logic" to mimic human thinking. This network was a binary classifier that could distinguish two different classes based on the input values. The problem of this network was the adjustment of weights by a human operator. After that, in 1957, the perceptron algorithm was proposed by Rosenblatt, which could learn the weights to classify the data in its structure without the participation of a human operator.

The first attempts at developing deep learning algorithms were made by Alexey Grigoryevich Ivakhnenko and Valentin Grigor'evich Lapa in 1965. They used models with polynomial activation functions (complex equations) that were then analyzed statistically. From each layer, the best-selected statistical features were transferred to the next layer.

While the perceptron method was used for several years, in 1969, Minsky and Papert published a paper presenting that the perceptron was more capable of

classifying linear problems, and this method could not solve non-linear problems. In addition, the authors of this article in the same year claimed that there are no sufficient computational resources to build large and deep neural networks.

The first "convolutional neural networks" were presented by Kunihiko Fukushima. Fukushima designed neural networks with multiple pooling and convolution layers, and in 1979, he developed an artificial neural network called the neocognitron that used a multilayered hierarchical design. This design allowed the computer to learn visual patterns. In addition, Fukushima's design allowed for the manual adjustment of critical features by increasing the "weight" of certain connections. In 1989, Yann LeCun presented the first practical demonstration of backpropagation at Bell Labs. He combined convolutional neural networks with backpropagation to read handwritten digits. This system was eventually used to read handwritten check numbers.

In the 1990s, some people continued to work on AI and DL, and significant progress was made. In 1995, Dana Cortes and Vladimir Vapnik developed the support vector machine (a system for mapping and recognizing similar data). Long short-term memory (LSTM) for recurrent neural networks was developed in 1997 by Sepp Hochreiter and Juergen Schmidhuber.

The next important evolutionary step for deep learning occurred in 1999 when computers became faster at processing data using GPUs (graphics processing units). Faster processing, with the GPU, increased the speed of calculations by 1000 times over 10 years. Neural networks also improved further with more available training data.

Around 2000, the vanishing gradient problem appeared. It was found that the "features" (lessons) that are formed in the lower layers do not learn anything from the upper layers because the lower layers receive no learning signal. Of course, this was not a fundamental problem for all neural networks and only happened to networks that used gradient-based learning methods. Two solutions used to solve this problem were layer-by-layer pre-training and the development of LSTMs.

Around 2006, deep belief networks (DBNs) and a layered pre-training framework were developed. During 2011 and 2012, AlexNet, a convolutional neural network, won many international competitions. The generative adversarial network (GAN) was introduced in 2014 by Ian Goodfellow. With GAN, two neural networks play against each other in the same game. The game's objective is for a network to mimic a photo and trick its opponent into believing it to be real. At the same time, the opponent is looking for its flaws. The game is played until an almost perfect picture deceives the opponent.

BERT, developed by Google in 2018, is a machine learning technique applied to natural language processors that aims to better understand the language we use daily. It analyzes all the words used in the search to understand the entire context and get the user's desired results. BERT is a system that uses transformers, a neural network architecture that analyzes all possible relationships between words in a sentence. In March 2019, Yoshua Bengio, Geoffrey Hinton, and Yann LeCun received the Turing Prize for their continuous efforts on conceptual and engineering advancements in deep neural networks. In 2020, when there was a pandemic, OpenAI

created an artificial intelligence algorithm called GPT-3 (Generative Pre-trained Transformer 3) that could produce human-like text, which is currently (Jun 2023) GPT-4 is the most advanced language model in the world.

Figure 2.1 shows the timeline of deep learning according to its important milestones.

2.2.2 AI, ML, and DL

Artificial intelligence is the science and engineering of building intelligent machines, specially computer systems, by reproducing human intelligence through learning, reasoning, and adaptation. Artificial intelligence uses intelligent agents that understand their environment and take actions that maximize their chances of success in achieving their goals.

Data mining understands and discovers new, previously unseen knowledge in the data. A simple definition of data mining is that data mining refers to the use of algorithms to extract patterns from data [2]. Deep learning is considered a subset of machine learning and artificial intelligence; therefore, deep learning can be considered an artificial intelligence function that mimics the data processing of the human brain. Deep learning also refers to learning methods from data, where computations are performed through multilayer neural networks. The term "deep" in deep learning refers to the concept of multiple levels or steps through which data is processed to build a data-driven model. It differs from standard machine learning in terms of performance when the volume of data increases. The global popularity and scope of applications of "deep learning" are increasing daily. Deep learning technology uses multiple layers to represent the abstraction of data to build computational models. While deep learning takes a long time to train a model due to numerous parameters, it takes a short time to run during testing compared to other machine learning algorithms [1].

Data science is a scientific discipline related to the study of generalizable knowledge extraction from data and all preprocessing and processing steps related to data, including collection, storage, cleaning, interpretation, analysis, visualization, validation, and decision-making based on data [3]. Data science uses artificial intelligence, machine learning, data mining, and other approaches such as evolutionary algorithms, operational research, statistics, etc.

Figure 2.2 shows the positions of artificial intelligence, machine learning, deep learning, and data science relative to each other.

One of the most important differences between deep learning and machine learning is the performance of the system based on increasing training samples. Deep learning will not yield good results if there are not enough training samples. On the other hand, machine learning can show good results even with a few samples. In addition, deep learning requires advanced hardware, whereas machine learning can be used with low-power hardware and computers. The key difference that shows the

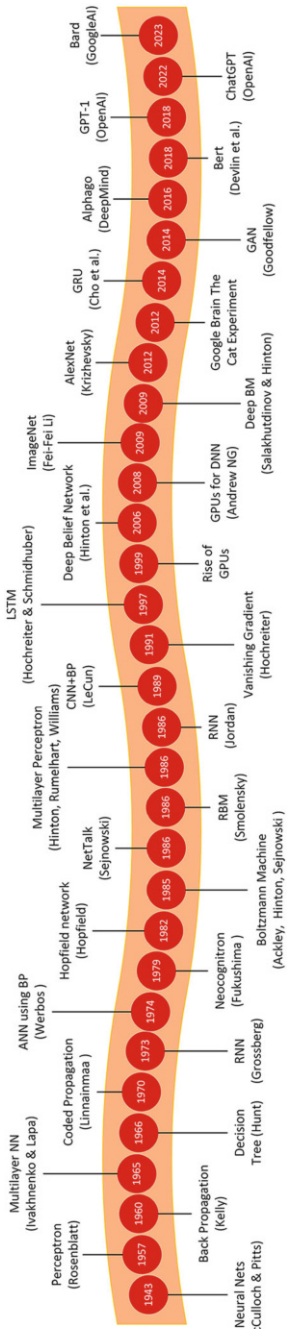

Fig. 2.1 Timeline of deep learning

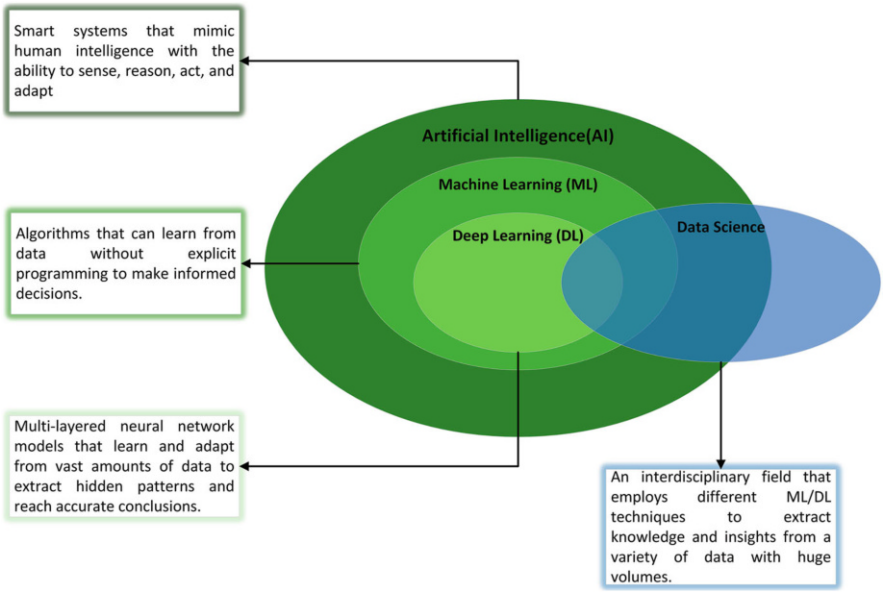

Fig. 2.2 The position of AI, ML, DL, and data science relative to each other

power of deep learning versus machine learning is the automatic extraction of features in these algorithms.

ML and DL are very similar in allowing the model to learn from previous data. The term ML can be generalized to any machine (model) that learns. DL is a specific set of methods and techniques that enable the machine to make decisions using very deep and complex networks. However, one of the most notable differences in DL is its ability to replace the human-based feature extraction process and incorporate this step into the neural network itself to automatically decide which features best describe the data (Fig. 2.3). Although DL models have proven to solve some of the most challenging problems, they can be data-hungry and computationally expensive. Before developing a DL-based solution, careful consideration of the hardware requirements for training and hosting complex DL models is required.

2.2.3 Advantages of Deep Learning

The deep learning process in systems is based on the construction of computational models called neural networks, which are inspired by the brain's structure. The structure of this network consists of several processing layers, which by going to the next level layers, it can solve more complex problems. The initial layers process the raw data, and the subsequent layers can use the information of the neurons in the previous layers to obtain a more complex representation of data. Most of the benefits

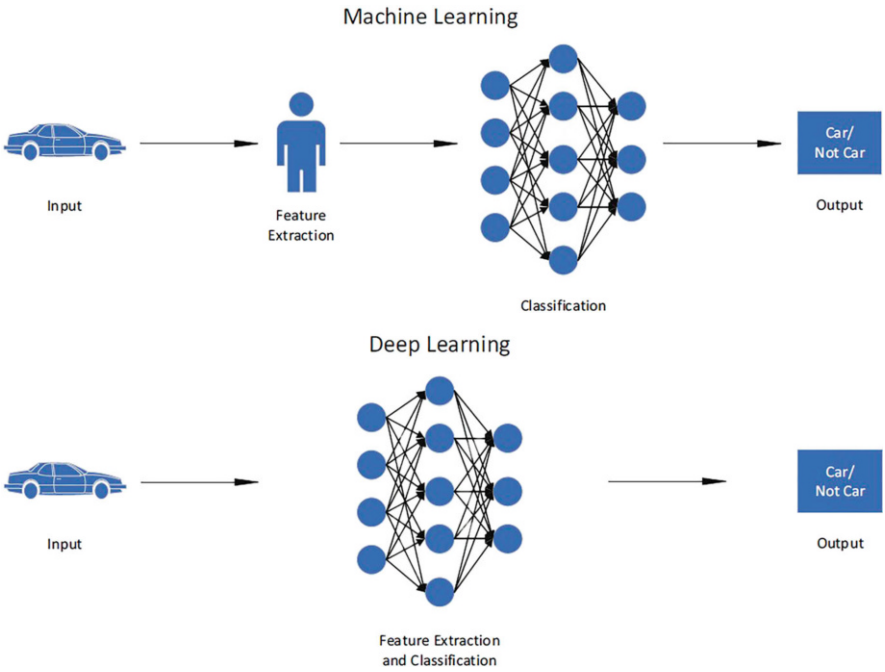

Fig. 2.3 Machine learning vs. deep learning [4]

of deep learning come from the fact that neural networks can learn to perform much better at feature extraction than any built artificial systems [5].

Briefly, the advantages of deep learning methods are as follows:

- **Automated feature extraction:** Deep learning algorithms can generate new feature representations from a limited number of features existing in the training dataset without additional human intervention. This means that deep learning can handle complex tasks requiring more extensive feature engineering and precision.
- **Ease of working with unstructured data:** One of the most prominent attractions of deep learning is its ability to work with unstructured data. The power of classical machine learning algorithms to analyze unstructured data is limited, but deep learning is most effective in handling this type of data.
- **Higher self-learning ability:** Multiple layers in deep neural networks allow models to learn complex features and perform computationally intensive tasks more efficiently. Moreover, deep learning algorithms can learn from their errors; in addition to confirming the correctness of their results, they can make the necessary adjustments. However, classical machine learning models require varying degrees of human intervention to determine output accuracy.
- **Support for parallel and distributed algorithms:** Parallel and distributed algorithms allow deep learning models to be trained much faster. Models can

be trained on machines equipped with high-performance CPUs, GPUs, or a combination of both.

- **Advanced analytics:** Deep learning, when applied to data science scope, can provide better and more effective processing models. Its ability to learn in an unsupervised mode continuously improves accuracy. It also provides data scientists with more reliable and concise analysis results.
- **Scalability:** Deep learning is highly scalable due to its ability to process massive amounts of data and perform many computations cost-effectively and cost-efficiently. This directly affects modularity, portability, and productivity.
- **Increased robustness:** Deep learning approaches do not need in-advanced designed features. Instead, they learn the optimal features automatically in the learning process. As a result, robustness is obtained with respect to changes in input data.
- **Generalization:** Deep learning approaches can be used in different applications or with different types of data.

Deep learning techniques in various fields, such as image processing, social network analysis, information retrieval, natural language processing, robotics, industrial automation, agriculture, medical research, disease diagnosis, recommender systems, motion detection systems, etc., can be used. In general, deep learning techniques are helpful in the following cases:

- Lack of human experts
- Learning skills that humans are unable to express and explain, such as understanding language, image, and sound
- The solution dynamism and its changes over time
- The large size of the problem compared to the limited inference capabilities of humans
- Problems with special constraints, such as biometrics

2.2.4 General Process of Deep Learning-Based Solutions

Each approach utilizing deep learning techniques includes different stages according to their deep learning model. However, they all generally follow the steps shown in Fig. 2.4. The datasets collected from various data sources in the data acquisition phase must be first preprocessed. The preprocessing steps are data cleaning, normalization, scaling, and quality assessment. After that, data transformation is performed to enhance the preprocessed in different stages of standardization, reduction, and aggregation. During this phase, feature engineering is also performed, and the resulting data representations are split into the training, testing, and validation sets. It should be mentioned again that, unlike machine learning methods, the process of feature extraction is done automatically in deep learning methods. After this step, and based on the nature of the problem and its requirements, the architecture of the deep learning approach, including discriminative, generative, graph, or

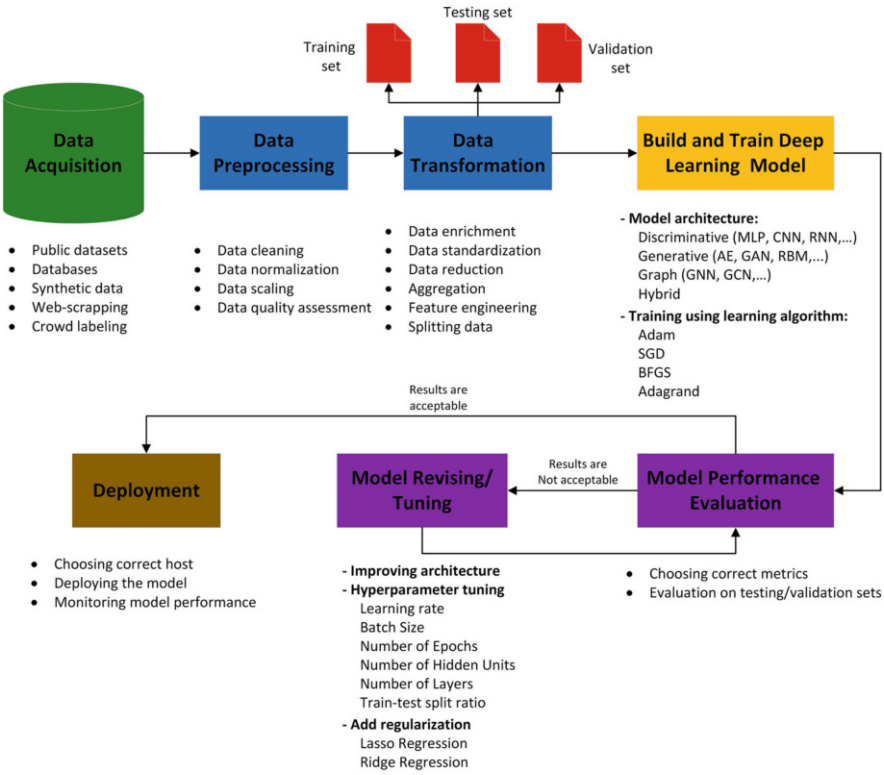

Fig. 2.4 Workflow of deep learning models

hybrid, is developed. During this phase, the type of learning algorithms, such as Adam, SGD, BFGS, etc., should be developed and evaluated. The built model is trained and then evaluated based on different evaluation metrics. If the obtained results are acceptable, the final model will be deployed on the target platform; otherwise, the model should be improved/revised/tuned, and the performance evaluation step should be repeated till acceptable results are achieved.

2.2.5 Taxonomy of Deep Learning Models

Deep learning models can be classified into three general groups, discriminative, generative, and graph methods, based on their nature, architecture, and performance [1]. Models based on discriminative methods usually specify the decision boundary in the data space, while models based on generative methods learn the overall distribution of the data [6]. Discriminative methods include convolutional neural networks (CNN), recurrent neural networks (RNN), and multilayer perceptron (MLP). The major generative methods include autoencoders (AE), generative

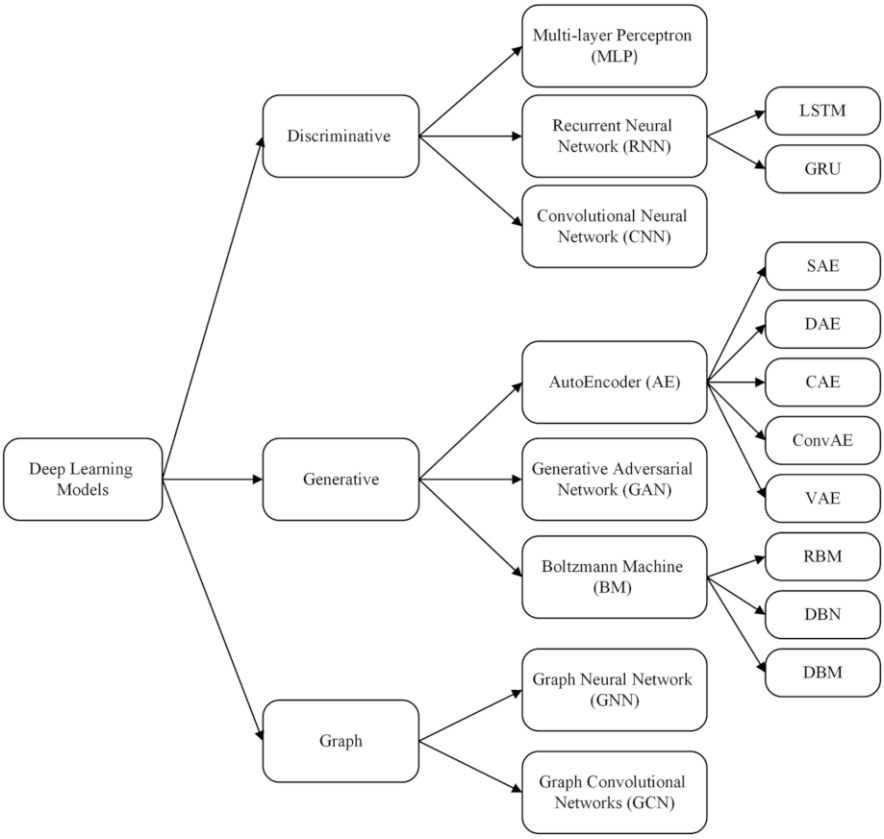

Fig. 2.5 The classification of deep learning models

adversarial networks (GAN), and different types of Boltzmann machine (BM) models. Graph-based methods also include graph neural networks (GNNs) and graph convolutional networks (GCNs). It should be mentioned that many hybrid models also result from various combinations of different models.

The taxonomy of deep learning models is shown in Fig. 2.5.

In the following of this chapter, a variety of deep discriminative models will be examined in Sect. 2.3, deep generative models in Sect. 2.4, and graph-based models in Sect. 2.5.

2.3 Deep Discriminative Models

Machine learning models often present the relations between features x and labels y using a joint probability p over x,y. According to the method of calculation of p, machine learning models are known as generative or discriminative [7]. In discriminative methods to predict y based on x, the conditional likelihood model $p(y|x)$ is fitted. Since discriminative models do not model $p(x)$, they may use their parameters more effectively to identify the probability $p(y|x)$. This makes them more suitable for supervised learning problems, and by making fewer modeling assumptions, they may use the data more efficiently [8]. A deep discriminative model uses layered hierarchical architectures to directly compute $p(y|x)$ [9]. This class of deep learning techniques is used to provide a discriminative function in supervised or classification applications. Deep discriminative architectures are usually designed to provide discriminative power for pattern classification by describing posterior distributions of categories conditioned on observable data.

Deep discriminative models generally include multilayer perceptron networks, recurrent neural networks, and convolutional neural networks, and each of these models will be discussed in the following subsections.

2.3.1 Multilayer Perceptron

The multilayer perceptron (MLP) neural network is a feedforward artificial neural network that is the basis of deep neural network (DNN) architecture [10]. MLP consists of an input layer to receive input signals and data, an output layer to predict or make decisions related to inputs, and, between these two layers, an arbitrary number of hidden layers that are responsible for the main function of the MLP. MLPs can approximate continuous functions using hidden layers. MLP neural networks are often applied to supervised learning problems. They are trained on a set of input-output pairs and learn to model the dependencies between inputs and outputs. The training phase consists of adjusting the parameters or weights and the biases of the model to minimize the error.

In MLP, the backpropagation algorithm is used to adjust the weight and the amount of bias toward the error, and its main purpose is to reduce the value of the loss function by adjusting the values of the weights and bias of the network. The backpropagation algorithm is the core of neural network training, which adjusts the weight of the neural network obtained in the previous epoch. This algorithm moves in two directions, forward and backward, in the network. It can calculate the error gradient value for any network parameter (any weight or bias). In this way, it can determine how much the value of each weight in an MLP neural network should change.

During the training process, various optimization approaches, such as stochastic gradient descent [11] (SGD), [12] BFGS (L-BFGS), and ADAM [13], can be

applied, which are algorithms that attempt to minimize the loss function. The gradient descent algorithm is an iterative method that tries to minimize the loss function by changing the internal weights of the network and gradually updating them. The step size in each iteration of the algorithm determines the learning rate, and the iteration process is carried out until there is no change in the loss function. In practice, when the number of training samples is large, using the gradient descent algorithm will take much time. This must be done in each iteration of the algorithm for all samples. For this reason, using the stochastic gradient descent algorithm will be more useful because it only updates a set of samples in each iteration of the algorithm.

Stochastic gradient descent is a stochastic approximation method of gradient descent in which each sample is randomly selected for optimization in each period, and new weights are obtained. But it may get stuck in the local minima, which is why the mini-batch gradient descent was presented, which divides the entire training set into mini-batches and updates the parameters based on these mini-batches [14]. This method is more resistant to noise and has less variance; as a result, it has more stable convergence due to the use and combination of full gradient reduction and stochastic gradient descent. Therefore, this optimization method is usually used in deep learning, but determining the learning rate is essential. Learning rates in other methods, such as ADAM, Adagrad [15], or Adadelta [16], are adjusted adaptively and do not need manual adjustment. The ADAM algorithm outperforms other adaptive methods and converges very quickly. It also overcomes other problems, such as learning rate decay, high variance in updating, and slow convergence.

The outputs of neurons in an MLP network are determined using various activation functions, also known as transfer functions. These functions use simple mathematical calculations to determine whether a node's input is important to the network or should be ignored. In other words, the activation function maps the sum value of the neuron's weighted input to values between 0 and 1 or -1 and 1 (depending on the type of activation function). Then, this function passes its final value to the next layer. For this reason, this function is also called the transfer function. There are three categories of activation functions: binary step, linear, and non-linear. The binary step function is compared to a threshold value. If the input value is greater than the threshold value, the node will be activated; otherwise, it will remain disabled, and the output of the node will not be passed to the next layer. This function cannot produce multiple-valued outputs and cannot be used for problems such as multi-class classification. Also, the derivative of the binary step function is equal to zero, which is a challenge for the backpropagation algorithm.

The linear activation function or the identity function does not perform computations on the weighted input sum and transfers this value to the next layer without any changes. This function cannot be used in the backpropagation algorithm because the derivative of this function is equal to a fixed number, has no relation with the input value x, and does not show good performance for complex neural networks with many parameters. On the other hand, the output of several linear functions is the same for a fixed input value. Therefore, it does not matter if the deep neural network is made of several hidden layers because the output of the activation function in the

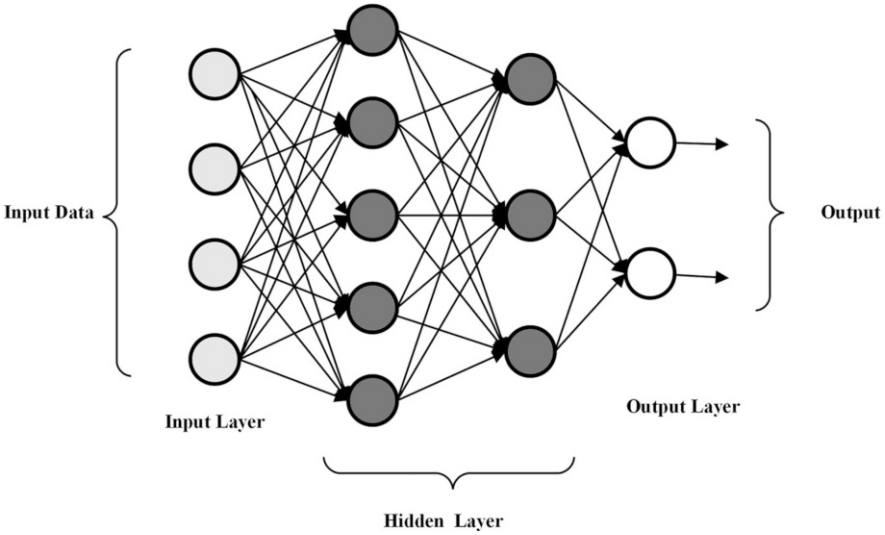

Fig. 2.6 The general architecture of MLPs

last layer of the neural network is equal to the output of the activation function in the first layer.

The non-linear activation functions are the most widely used in neural networks because the generalizability and adaptability of the model to different types of data are made easy by using these functions. These functions solved the problem related to the backpropagation algorithm and can determine which input node weight can better contribute to the final diagnosis of the model. Using these functions, you can also solve problems related to multiple outputs. There are various types of non-linear activation functions, such as [17]:

Sigmoid, hyperbolic tangent (tanh), rectified linear unit (ReLU), leaky rectified linear unit (leaky ReLU), parametric rectified linear unit (parametric ReLU), exponential linear unit (ELU), softmax, Swish, Gaussian error linear unit (GELU), and scaled exponential linear unit (SELU).

Each of these functions has its own characteristics. To choose the most suitable activation function for the final layer of the deep neural network, one should pay attention to the purpose of the model and the type of prediction of the model. MLP requires setting several hyperparameters, such as the number of hidden layers, neurons, and iterations, which can make solving a complex model computationally expensive. However, MLP provides the advantage of learning non-linear models in real time or online through relative fitting. Figure 2.6 shows the general architecture of the MLP.

2.3.2 Convolutional Neural Network

Convolutional neural network (CNN or ConvNet) is a discriminative deep learning architecture that learns features directly from the input without the need for human feature extraction [18]. CNNs are widely used in various domains, such as image processing, natural language processing, speech processing, etc. For some types of data, specially images, methods such as MLP do not work well because each neuron is fully connected to each neuron in the next layer and each neuron in the hidden layer computes a function that depends on the values of the nodes in the input layer. However, in CNNs, only a local subset of the previous layer variables is considered. CNNs are more similar to the human visual processing system than conventional neural networks; they perform more effective optimization for processing and learning of 2D and 3D images and extract input features automatically [19]. CNNs use local connections and shared weights in the network to extract features of the input data, which results in much fewer parameters and makes training the network faster and easier. This action is similar to the activity performed in the visual cortex cells. These cells are sensitive to small parts of the scene rather than the whole scene. In other words, the cells act as local filters on the input and extract the local correlation in the data.

The CNN learning process is divided into two general phases: feature engineering/learning and classification based on fully connected layers. To extract and learn features, there are usually several convolution layers followed by pooling layers, and in the final stage, fully connected layers (MLP) are employed. The output nodes of convolution and pooling layers are grouped in a two-dimensional plane called a feature map. The nodes of a plane are connected to a small area of each connected plane of the previous layer. Each node of the convolution layer extracts features from the input images by convolution operations on the input nodes.

The main task of the convolution layer is to detect features in local regions of input common to the entire dataset. Using filters to detect features leads to the production of a feature map. The pooling layer is used periodically between two successive convolutional layers, and its task is to reduce the dimensions of the feature map. In addition to extracting important features in the feature map, this work reduces the computing capability required for data processing by reducing the number of parameters. Pooling layers are two types: max pooling and average pooling. Maximum pooling (or max pooling) calculates the maximum value for each patch on the feature map, and average pooling calculates the average value for each patch on the feature map. After using several different layers, the fully connected layer at the end of the CNN network can be used to calculate desired features and output scores. A fully connected layer in CNN works like a hidden layer in MLP and performs a classification. Figure 2.7 shows the general architecture of the CNN model.

If the input x for CNN is considered as three-dimensional $m \times m \times r$, m is the height and width of the input, and r is the depth or the number of channels. In each convolution layer, there are k filters (kernels) of size $n \times n \times q$. Here, n must be

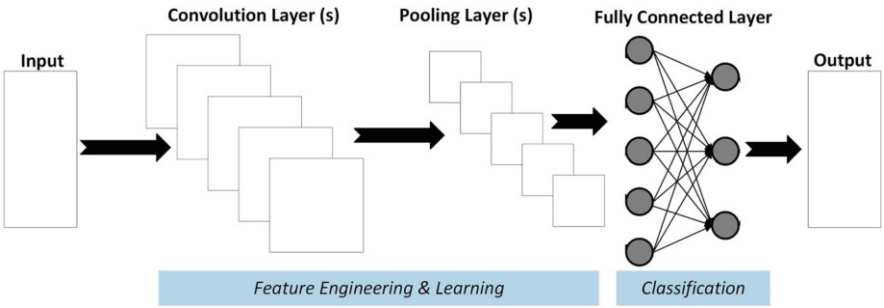

Fig. 2.7 The general architecture of CNNs

smaller than the input m, but q can be smaller or equal to r. The filters are the basis of the local connections, which share a similar bias (b^k) and weight W^k parameters for generating k feature maps (h^k) (the size of each of the feature maps is $m-n-1$).

As shown in Eq. (2.1), the convolution layer calculates a dot product between the weights and its inputs, and the inputs are small regions of the original input volume. Then, a non-linear activation function f is applied to the output of the convolution layers:

$$h^k = f\left(W^k * x + b^k \right) \tag{2.1}$$

After that, in the subsampling layers, the number of samples of each feature map is decreased to reduce the parameters in the network, speed up the training process, and control overfitting. A pooling operation (e.g., average or maximum) is performed on an adjacent $p \times p$ region (where p is the filter size) for all feature maps. Finally, the layers of the final stage, which are fully connected, take the previous low/middle-level features and generate a high-level abstraction of the data. The last layer (e.g., softmax) can be used to generate classification scores, where each score is the probability of a particular class for a given sample.

Softmax is an activation function that scales numbers/logits into probabilities. The output of a softmax is a vector that specifies the probabilities of each possible outcome or classes and the sum of probabilities in this vector is equal to one. Mathematically, softmax is defined as Eq. (2.2):

$$\text{Softmax}\ (y_i) = \frac{\exp(y_i)}{\sum_{j=1}^{c} \exp(y_j)} \tag{2.2}$$

where Y represents the values from the neurons of the output layer and c is the number of classes. The exponential acts as the non-linear function. Later, these values are divided by the sum of exponential values in order to normalize and then convert them into probabilities. It is worth mentioning that the softmax layer must have the same number of nodes as the output layer.

CNN network parameters are learned by the backpropagation algorithm and stochastic gradient descent algorithm optimization. The first phase is forward propagation, where signals are propagated from the input to the output of the network. In the last layer, the output of the cost function is compared with the actual value, and error estimation is performed. In the second phase, the backpropagation algorithm is used again to compensate for this error. However, the learning process in CNN is more complicated compared to the MLP neural network because it consists of different types of layers and the forward and backward propagation phases follow special rules in each layer. Neurons in CNN have a common weight, unlike MLP, where each neuron has a separate weight vector. This sharing of weights reduces the total number of trainable weights.

In general, most deep convolutional neural networks are built on a key set of essential layers, including convolution, subsampling, and fully connected layers. Specific architectures typically consist of stacks of multiple convolution layers, pooling layers, fully connected layers, and softmax layers in the network. Some examples of these models are LeNet [18], AlexNet [19], VGGNet [20], NiN [21], and All-CNN [22]. Other types and more efficient advanced architectures have been proposed in recent years, including DenseNet [23], FractalNet [24], GoogLeNet with inception units [25], and ResNets with residual layers [26]. The main components of the structure (convolution and pooling) are almost identical in these architectures. However, some topological differences are observed in modern deep learning architectures.

It should be mentioned that deep CNN, AlexNet [19], VGGNet [20], GoogLeNet [25], DenseNet [23], and FractalNet [24] architectures are generally more popular than other architectures due to their success in object detection on various datasets. Among all these architectures, some architectures, such as GoogLeNet and ResNet, are specially designed for large-scale data analysis, while the VGG network is considered a general architecture.

2.3.3 Recurrent Neural Network

Many of the data in the world are order-based and considered sequential, such as a user's transactions on an online sales Web site, stock prices in the stock market, and movies viewed by a user on Netflix. To process this type of data, deep learning methods should be used that can model the dependencies between data. Standard neural networks and CNNs cannot process this data because they only take a fixed-size vector as input and produce a fixed-size output. Second, these models operate with a fixed number of computational steps (e.g., the number of model layers) [27]. However, recurrent neural networks are unique because they can operate on a sequence of input vectors over time. These networks remember the dependencies between sequential data using hidden states (or memory) and do not consider the data independent of each other [28].

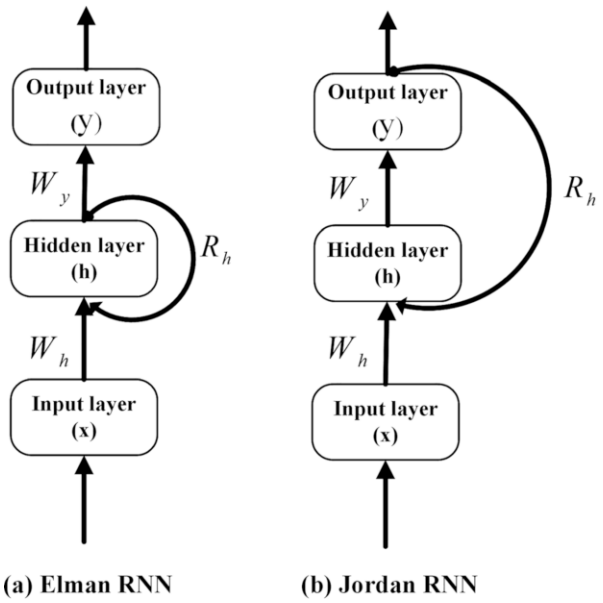

Fig. 2.8 RNN schematics proposed by Elman and Jordan

The main characteristic of RNN is that the recurrent units have hidden states that are not only dependent on the current input of the network but also related to the previous inputs. Different versions of RNN have been proposed by Jordan and Elman [29, 30]. In Elman, the architecture uses the output of the hidden layers as its input in addition to the normal inputs of the hidden layers. On the other hand, the outputs of the output unit are used as its inputs to the hidden layer in the Jordan network. RNN element model computations in the Elman model are according to Eqs. (2.3) and (2.4):

$$h_t = \sigma_h(W_h x_t + R_h h_{t-1} + b_h) \tag{2.3}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{2.4}$$

Jordan RNN model computations are also according to Eqs. (2.5) and (2.6):

$$h_t = \sigma_h(W_h x_t + R_h y_{t-1} + b_h) \tag{2.5}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{2.6}$$

In Eqs. (2.3), (2.4), (2.5), and (2.6), the parameter x_t is the input vector, h_t is the hidden layer vector, y_t is the output vector, W and R are the weight matrices, and b is the bias vector. The difference between these models lies in the position of the loop connection giving the recurrent property to the network. The high-level schematics of Elman and Jordan models have been depicted in Fig. 2.8.

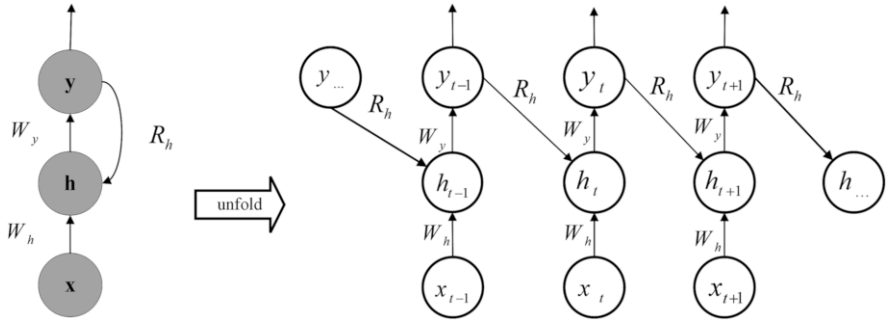

Fig. 2.9 The general architecture of RNNs

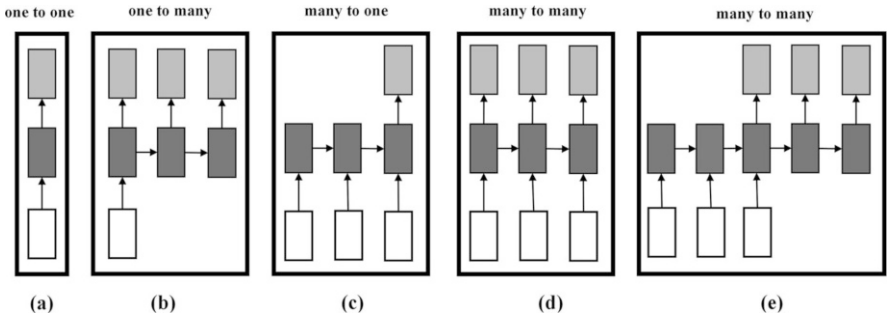

Fig. 2.10 Various architectures of RNN networks

Concluding from the above descriptions of the primary models of RNN, the general architecture of RNNs is shown in Fig. 2.9. As mentioned in RNN, each state depends on all previous computations by a recursive equation. An important effect of this is to create memory over time, as states are based on previous steps.

The various architectures for RNNs are categorized as one-to-one, one-to-many, many-to-one, and many-to-many [31], as shown in Fig. 2.10. In the one-to-one architecture, an RNN input unit is mapped to a hidden unit and an output unit (Fig. 2.10a). In the one-to-many architecture, one input unit of RNN is mapped to several hidden units and several output units, which is an example of image annotation. The input layer receives an image and maps it to multiple words (Fig. 2.10b). In the many-to-one architecture, several RNN input units are mapped to several hidden units and one output unit. A practical example of this architecture is emotion classification, where the input layer receives multiple tokens from different words of a sentence and maps them into a positive or negative polarity (Fig. 2.10c). In the many-to-many architecture, several input units of the RNN are mapped to several hidden units and several output units. A practical example of this architecture is machine translation, where the input layer receives multiple tokens of source language words and maps them to tokens of words in the target language (Fig. 2.10d, e).

From a theoretical point of view, RNNs can remember information for a long time, but in practice, they only see a few previous steps and do not have long-term memory. The standard RNN can hardly extract the long-term dependencies of the data due to vanishing and exploding gradients. Indeed, RNNs suffer from the problem of preserving the context for long-range sequences (think long sentences or long speeches). The effect of a given input on the hidden layer (and thus the output) either decays exponentially (vanishes) or blows and saturates (explodes) as a function of time (or sequence length).

Several solutions to solve this problem of RNNs have been proposed in the past few decades. Two possible effective solutions for this problem are, firstly, clipping the gradient and scaling the gradient if the norm is too large and, secondly, developing a better RNN model. Therefore, modified RNNs such as GRU and LSTM have been proposed, which have solved these problems by adding a gate function to the RNN network. In the following subsections, we briefly review GRU and LSTM models.

2.3.3.1 LSTM

One of the improved types of RNN networks is the long short-term memory (LSTM) model. LSTM was introduced to reduce the vanishing gradient problem and has become one of the most popular RNN architectures to date [32]. The standard LSTM has three gates, the forget gate f_t, which specifies how much of the previous data is to be forgotten; the input gate i_t, which evaluates the data to be stored in memory; and the output gate o_t, which decides how to calculate the output based on the available data and information, calculated by the below equations:

$$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i) \tag{2.7}$$

$$f_t = \sigma\left(W_f x_t + R_f h_{t-1} + b_f\right) \tag{2.8}$$

$$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o) \tag{2.9}$$

In Eqs. (2.7), (2.8), and (2.9), σ usually represents a sigmoid function, parameters W and R are the weight matrices, and b is the bias vector that can be trained. LSTM units are defined based on Eqs. (2.10)–(2.13):

$$\acute{\mathrm{C}}_t = \tan h(W_c x_t + R_c h_{t-1} + b_c) \tag{2.10}$$

$$C_t = \left(f_t \odot \mathrm{C_{t-1}} + \mathrm{i_t} \odot \acute{\mathrm{C}}_\mathrm{t}\right) \tag{2.11}$$

$$h_t = (o_t \odot \tan \mathrm{h}(\mathrm{C_t})) \tag{2.12}$$

$$y_t = \sigma\left(W_y h_t + b_y\right) \tag{2.13}$$

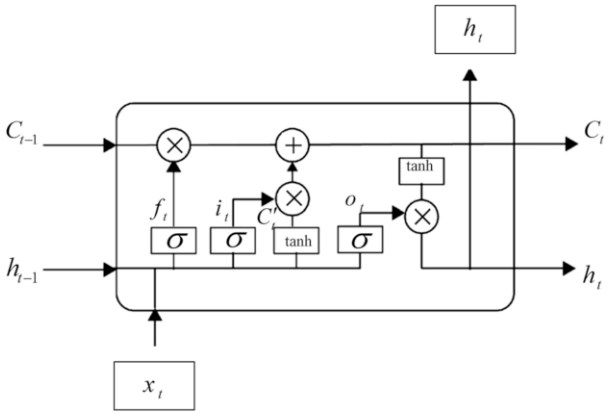

Fig. 2.11 The internal structure of LSTM cell

In fact, the state of the candidate cell $\acute{C}_t$ is calculated based on the input data x_t and the previous hidden state h_{t-1}. Memory or current cell state C_t is obtained by using forget gate f_t, previous cell state C_{t-1}, input gate i_t, and candidate cell state $\acute{C}_t$. The sign $\odot$ means to use element-wise multiplication. The output y_t is calculated based on the weights (W_y and b_y) corresponding to the hidden state h_t. Figure 2.11 shows the internal structure of the LSTM cell.

Over time, various types of LSTMs were developed in different research, such as stacked LSTM [33, 34], bidirectional LSTM [35], convolutional LSTM [36], multi-dimensional LSTM [37], graph LSTM [38], etc.:

- **Stacked LSTM:** In typical applications, the simplest method of increasing network capacity and depth in the LSTM network is to stack LSTM layers [33, 34]. A stacked LSTM network is the most basic and simplest LSTM network structure, which can also be considered as a multilayer fully connected structure.
- **Bidirectional LSTM:** Normal RNNs can only use the previous context. To overcome this problem, bidirectional RNN (B-RNN) was introduced by Schuster and Paliwal (1997) [39]. This type of architecture can be trained in both temporal directions simultaneously, with separate hidden layers (i.e., forward and backward layers). Therefore, to propose bidirectional LSTM in [35], Graves and Schmidhuber combined the B-RNN method with the LSTM cell and proposed B-LSTM.
- **Convolutional LSTM:** The fully connected LSTM layer contains too much redundancy for spatial data. Therefore, to perform a spatiotemporal sequence prediction problem, convolutional LSTM (ConvLSTM) was proposed by Sainath et al. [36], which uses convolutional structures in recurrent connections. The

ConvLSTM network uses the convolution operator to compute the next state of a particular cell, and then the next state is determined by the inputs and past states of its local neighbors.

- **Multi-dimensional LSTM:** Standard RNNs can only be used to deal with one-dimensional data. Multi-dimensional LSTM for expanding the application scope of RNNs was introduced by Graves et al. [37]. Its main idea was to create recursive connections as large as the dimensions of the data. At each point in the data sequence, the iterative layer L receives both the output of layer $L-1$ and its activations from one step back in all dimensions. This means that the LSTM cells in layer L have n forget gates in the n-dimensional LSTM network.
- **Graph LSTM:** An extended fixed topology of LSTM has been proposed by Liang et al. to develop the graph LSTM network based on the graph RNN network [38]. The LSTM graph model assumes that each superpixel node is defined by its previous states and adaptive neighboring nodes. In this method, instead of using a fixed starting node and a predefined update path for all images, the starting node and the updating scheme of LSTM graph nodes are determined dynamically.

2.3.3.2 GRU

LSTM can learn better than the standard RNN. However, additional parameters increase the computational complexity and load [40]. Therefore, gated recurrent unit (GRU) was introduced by Chu et al. in 2014 [41]. GRUs alleviate the vanishing gradient problem by using a mechanism similar to LSTM. GRUs are simpler than LSTMs because they use one less gate and eliminate the need to distinguish between hidden states and memory cells. This type of mechanism is widely used and popular due to the simplicity of the model and reduced complexity and computational costs compared to LSTMs. In this type of network, forget and input gates are combined to create an update gate. On the other hand, cell and hidden states are also merged.

GRU has two update gates u_t and reset gate r_t. The u_t gate sets the update rate of the hidden state, and the r_t gate decides how much past information is to be forgotten. Equations (2.14)–(2.18) show the formulation of GRU:

$$u_t = \sigma(W_u x_t + R_u h_{t-1} + b_u) \tag{2.14}$$

$$u_t = \sigma(W_u x_t + R_u h_{t-1} + b_u) \tag{2.15}$$

$$h'_t = \tan h(W_h x_t + (r_t \odot h_{t-1}) R_h + b_h) \tag{2.16}$$

$$h_t = ((1 - u_t) \odot h_{t-1} + u_t \odot h'_t) \tag{2.17}$$

$$y_t = \sigma(W_y h_t + b_y) \tag{2.18}$$

Figure 2.12 shows the internal structure of the GRU cell.

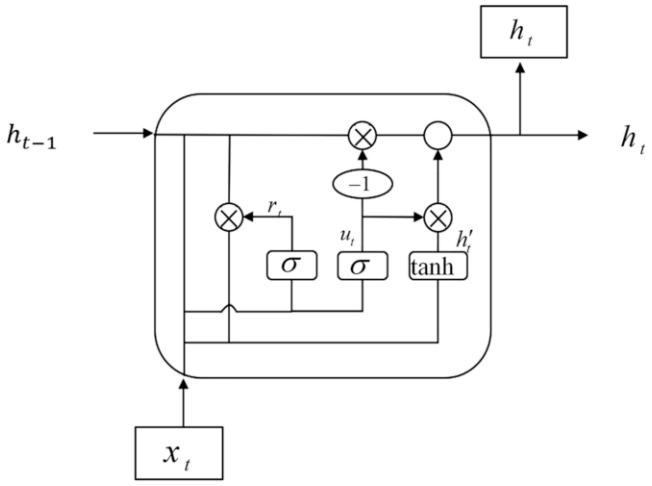

Fig. 2.12 The internal structure of GRU cell

Several types of GRU have been developed in different research, such as bidirectional GRU, stacked bidirectional GRU, convolutional GRU, etc. [42]:

- **Bidirectional GRU:** A bidirectional GRU or BiGRU is a sequential processing model consisting of two GRUs. One takes the input in the forward direction and the other in the backward direction. In BiGRU, both layers are independent; however, they have the same input sequence, and the final outputs of the two layers are connected. The forward layer reads the input sequence from left to right, and the backward layer reads the input sequence in reverse order from right to left. Each cell consists of two gates, reset and update, with two activation functions.
- **Stacked bidirectional GRU (stacked BiGRU):** When different layers of BiGRU are stacked next to each other, a stacked bidirectional GRU is created. The stacked BiGRU was used to encrypt the information to get more detailed information and features. Some data, like sentences, are directed, and stacked BiGRU is one of the suitable options for processing these types of datasets.
- **Convolutional GRU:** This model is a type of GRU that combines GRUs with convolution operations. Specifically, CNN-GRU extracts features through the convolution layer and performs time series prediction by stacking multiple GRU layers. Similar to other deep neural network models, the CNN-GRU training method is implemented using backpropagation and gradient descent. The goal of the training process is to reduce the root mean square error.

2.4 Deep Generative Models

Generative models are considered a type of deep learning model whose goal is to learn how to generate new samples from the same training dataset [43]. During the training phase, a generative model attempts to solve a density estimation problem. In density estimation, the model learns to make an estimate as close as possible to the unobserved probability density function. Furthermore, the generative model should be able to create new distribution samples in addition to processing existing samples. Generative models should recognize the distributions and basic features of the data to reconstruct or generate similar samples and learn them efficiently. A model capable of generating new samples can be said to have learned and comprehended a concept without training. For this reason, these models are classified as unsupervised models.

Deep generative models are neural networks with many hidden layers that are trained for complex estimation and high-dimensional probability distributions [44]. The most important goal of training this type of model is to learn intractable or unknown statistical distributions from a few independent samples that are uniformly distributed. After successful training, deep generative methods can be used to estimate the likelihood of a specific sample and create new samples similar to the unknown distribution.

Deep generative learning models are used by researchers in various domains due to their high flexibility in statistical distributions and their high capacity to learn non-linear representations. The most important examples of generative models are autoencoders (AE), generative adversarial networks (GAN), and models based on Boltzmann machines such as restricted Boltzmann machines (RBM), deep belief networks (DBN), and deep Boltzmann machines (DBM). Each of these models is described below in separate subsections.

2.4.1 Autoencoders

An autoencoder is an unsupervised deep learning method that learns how to efficiently compress and encode data and then reconstructs the data from a reduced encoded representation to one that is similar to the original input [45]. Autoencoders provide a method for automatically learning features from unlabeled data, which enables unsupervised learning. This neural network model applies backpropagation and sets the target values (outputs) equal to the inputs. Recently, the theoretical bridge between autoencoders and latent variable models has transferred autoencoders to the forefront of generative modeling.

In the autoencoder, in addition to the input, a layer with lower dimensions than the input and output is considered in the middle of the structure, which forces the autoencoder not only to transfer the input into the output but also to create a compressed version of the input in hidden layers which is called representation or

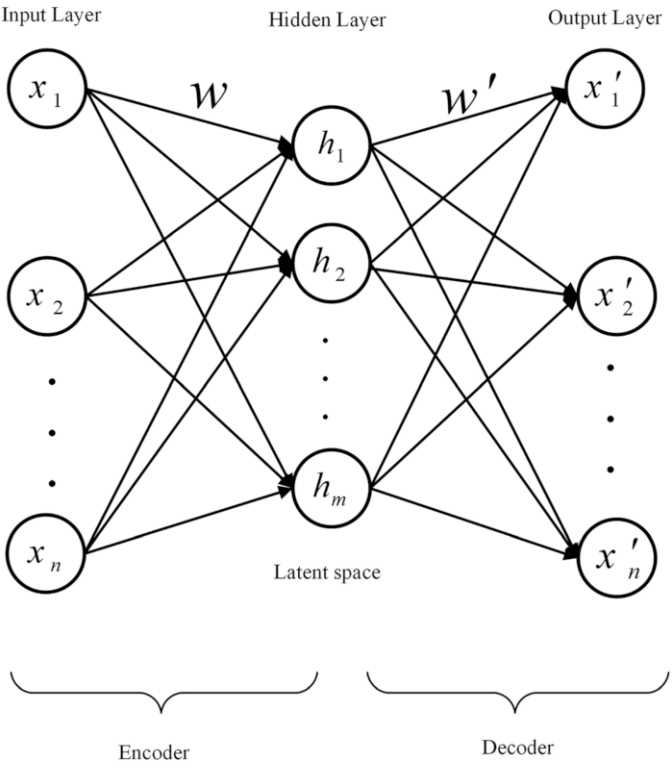

Fig. 2.13 The general architecture of autoencoders

code. The autoencoder includes layers such as the input, hidden, and output layers. Combining the input layer and the hidden layer creates an encoder, and combining the hidden layer and the output layer creates a decoder. The encoder compresses the input and generates the code, and the decoder reconstructs the input based on the code. Encoder and decoder are feedforward neural networks placed symmetrically in the autoencoder structure in most cases. The hidden layer is a layer with the appropriate dimensions, according to the designer. Notably, the number of neurons in the hidden layer is a hyperparameter. Figure 2.13 shows the general architecture of an autoencoder.

In the autoencoder, the input x is encoded in a low-dimensional space and then decoded by reconstructing $\acute{x}$ from the corresponding input [46]. Assuming a hidden layer, the encoding and decoding processes of an autoencoder are shown in Eqs. (2.18) and (2.19), respectively. The encoding and decoding weights are denoted as W and $W^{'}$, and the objective is to minimize the reconstruction error. $x = \{x_1, x_2, \ldots, x_n\}$ is the input of the autoencoder with high dimensions, and its encrypted representation is considered as $h = \{h_1, h_2, \ldots, h_d\}$. Equation (2.19) shows the conversion of the input to the encoded representation:

$$h = f(Wx + b) \tag{2.19}$$

In Eq. (2.19), f is the activation function, W is the weight matrix, and b is the bias vector. The decoder reconstructs the encrypted representation of the hidden layer and reaches data $\acute{x} = \{\acute{x}_1, \acute{x}_2, \ldots, \acute{x}_n\}$ using the g function. This decoder function is calculated using Eq. (2.20):

$$\acute{x} = g\left(\acute{W}h + \acute{b}\right) \tag{2.20}$$

In the above equation, g is the activation function, $W^{'}$ is the weight matrix, and b is the bias vector. Functions f and g are usually non-linear activation functions such as the tanh and sigmoid that help the autoencoder to learn more important and useful features than the PCA method by minimizing the reconstruction error between x and $\acute{x}$ and obtaining the d-dimensional representation of the input data.

To train an autoencoder, several parameters that affect the performance of the model must be set beforehand. These parameters include the type of optimizer, dropout, hidden layer size, number of layers, and cost function [47]. A type of optimizer that has a high speed and can handle high-volume data with low memory consumption should be selected. Dropout is also only used during training and then automatically disabled during execution. The size of the hidden layer is the number of nodes in the middle or hidden layer, and the number of layers determines the size of the required encoder and decoder layers. Autoencoders can be divided into deep and shallow networks depending on the number of hidden layers. The number of nodes in these layers should also be determined. The cost function evaluates the neural network training process. The cost function in the autoencoder is the mean square error or cross-entropy. Different types of autoencoders have been presented in research, which include sparse autoencoder [48], denoising autoencoder [49], contractive autoencoder [47], convolutional autoencoder [50], and variational autoencoder [51], which are briefly explained in the following subsections.

2.4.1.1 Sparse Autoencoder

The purpose of sparse autoencoders is to extract sparse features from raw data. Sparsity can be obtained either by penalizing hidden unit biases or directly by penalizing the output of hidden unit values. In this type of neural network, the number of hidden layer cells is greater than the number of input/output layer cells. Sparse representations have several benefits, including (1) the use of representations with large dimensions increases the probability that different categories can be separated easily, (2) sparse representations provide a simple interpretation of complex input data, and (3) biological vision uses sparse representations in primary visual areas.

The idea of this type of autoencoder is that a neuron is activated only for some training samples. Since the samples have different features, the activation of neurons

should not be done by the same method. The goal is a latent representation in which many elements in the representation are zero so that the most salient features are shown.

If $g(h)$ is the decoder output, $h = f(x)$ is the encoder output, and $\varepsilon(h)$ is the sparsity penalty, then the loss function of the sparse autoencoder is as follows:

$$L(x, g(f(x))) + \varepsilon(h) \tag{2.21}$$

In Eq. (2.21), sparsity penalty $\varepsilon(h)$ is based on the following logarithmic function:

$$\varepsilon(h) = \sum_{j=1}^{d} KL\left(p \middle\| \acute{p}_j\right) \tag{2.22}$$

In Eq. (2.22), $KL\left(p \parallel \acute{p}_j\right)$ is the Kullback-Leibler (KL) divergence between a Bernoulli random variable with mean p and a Bernoulli random variable with mean $\acute{p}_j$, and d is the number of neurons in the hidden layer.

The general architecture of the sparse autoencoder is shown in Fig. 2.14.

2.4.1.2 Denoising Autoencoder

In the denoising autoencoder, instead of adding a penalty to the loss function, it is possible to train the autoencoder with useful information by changing the reconstruction error of the loss function. This can be done by intentionally adding some noise to the input layer. By entering these noise values, the denoising autoencoder creates a corrupted copy of the input. A denoising autoencoder tries to improve the representation (to extract useful features) by changing the reconstruction measure. In other words, corrupted data are received as inputs and trained to recover the input without distortion and the original error as outputs. This is done by minimizing the average reconstruction error on the training data, i.e., removing the corrupted input or removing the noise. The input in this network is a corrupted version $\tilde{x} \in R^n$ of the original input $x \in R^n$. This autoencoder does not simply copy the input to the output but denoises the data and then constructs the input from the corrupted version. According to Eq. (2.23), this autoencoder minimizes the error on the corrupted input as follows:

$$L(x, g(f(\tilde{x}))) \tag{2.23}$$

In Eq. (2.23), $g(f(\tilde{x}))$ is the output of the decoder, and $f(\tilde{x})$ is the encoded output of the corrupted input. Therefore, in the field of computing, denoising autoencoders can be considered potent filters that can be used for automatic preprocessing. For example, an automatic denoising encoder can be used to automatically preprocess an image, thereby increasing its quality for an accurate detection process. The general architecture of this type of autoencoder is shown in Fig. 2.15.

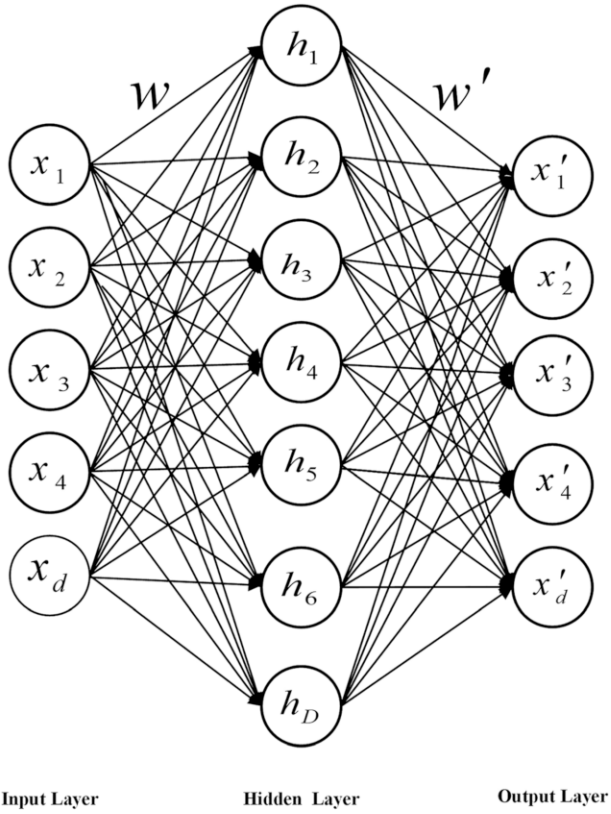

Fig. 2.14 The general architecture of sparse autoencoders

2.4.1.3 Contractive Autoencoder

This type of autoencoder is the further development of the denoising autoencoder, and both motivations are the same for robust learning of data representations. While the denoising autoencoder strength is the mapping operations by injecting noise into the training set, the contractive autoencoder achieves this by adding an analytic contractive penalty to the reconstruction error function. A denoising autoencoder with small corruption noise can be considered as a type of contractive autoencoder where the contractive penalty is on the whole reconstruction function instead of the encoder. Both the contractive and denoising autoencoder were successfully used in

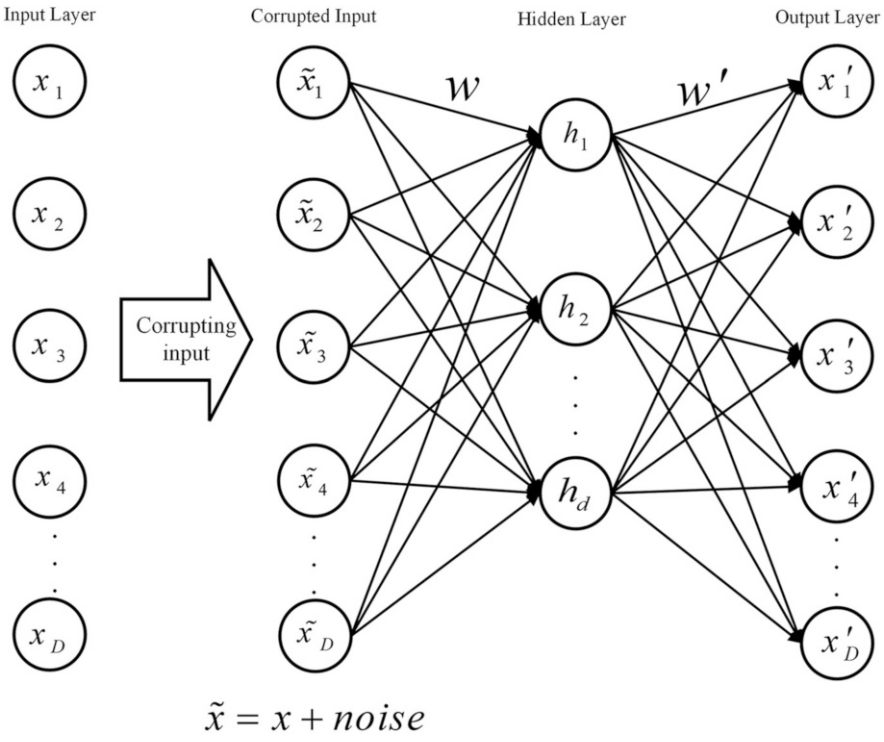

Fig. 2.15 The general architecture of denoising autoencoders

the transfer learning competition in an unsupervised mode. This is achieved by adding a penalty to the loss function, as shown in Eq. (2.24):

$$L(x, g(f(x))) + \varepsilon(h) \tag{2.24}$$

In the above equation, $g(f(x))$ is the output of the decoder, $f(x)$ is the output of the encoder, and $\varepsilon(h)$ is the sum of the square elements of the Jacobian matrix. In fact, this penalty is the sum of the squared elements of the Jacobian matrix of the partial derivatives of the encoder function, which is computed according to Eq. (2.25):

$$\varepsilon(h) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2 \tag{2.25}$$

In Eq. (2.25), the parameter λ is a hyperparameter used to control the regularization strength. The final result is a reduction in the sensitivity of the learned representation to the training input.

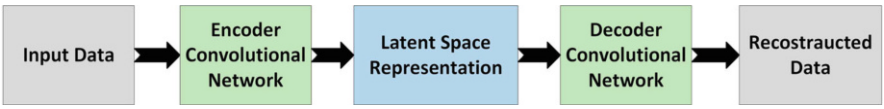

Fig. 2.16 The general architecture of convolutional autoencoders

2.4.1.4 Convolutional Autoencoder

The convolutional autoencoder extends the basic structure of the simple autoencoder by changing the fully connected hidden layers into convolutional layers. Similar to the simple autoencoder, the size of the input layer is the same as the output layer, but the encoder network is changed to convolution layers and the decoder network to transposed convolution layers (deconvolution). To extract the structural features of multi-dimensional data such as images, convolutional neural networks provide a better architecture. In addition, they can be stacked such that each convolutional autoencoder takes the latent representation of the previous convolutional autoencoder for higher-level representations. The difficult part of this autoencoder is on the decoder side of the model. During encoding, data sizes are reduced by subsampling with average or maximum pooling. Both operations result in data loss that is difficult to recover during decoding. The convolutional autoencoder allows the model to learn optimal filters to minimize the reconstruction error. Once these filters are learned, they can be applied to any input to extract features. Therefore, these features can be used to perform any task that requires a compressed representation of the input. Figure 2.16 shows the general architecture of this type of autoencoder.

2.4.1.5 Variational Autoencoder

A variational autoencoder is a type of autoencoder with additional constraints on the encoded representations being learned. More precisely, this autoencoder learns a latent variable model for its input data. Therefore, instead of the neural network learning an arbitrary function, in the variational autoencoder, the parameters of the probability distribution model its own data. If the points of this distribution are sampled, it produces new input data samples. Because of this, variational autoencoders are considered generative models.

Variational autoencoders attempt to decode encodings that come from a known probability distribution to produce appropriate outputs, even if they are not real data encodings. In variational autoencoders, instead of mapping the input to a fixed vector, the input is mapped to a distribution. Therefore, the important difference between this type of autoencoder and other types is that the bottleneck (hidden) vector is replaced by the mean vector and the standard deviation vector and then the sampled latent vector is considered as the real bottleneck. This is very different from the conventional autoencoder, where the input directly generates a latent vector.

Based on these features, the major applications of variational autoencoders are to reduce data dimensions and learn representations from them.

The function of this autoencoder is to take the input to provide two vectors of size n: the mean vector and the standard deviation vector. By using mean and standard deviation, we can generate samples with a normal distribution that corresponds to Eq. (2.26):

$$F(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2.26}$$

In Eq. (2.26), μ and σ are the mean and standard deviation for a random number X_i. After that, the obtained encoded data are passed to the decoder. New samples added to the distribution expand the space for further sample generation. On the other hand, the decoder learns that the samples are not generated from a single point but follow a distribution in the continuous latent space. During decoding, each latent feature is sampled to generate a sample vector and then sent to the decoder. In this way, a slight change in the input leads to an accurate reconstruction of the output. Statistically, this method uses Bayesian theory. If z samples are generated from x observations, the probability of z given x is calculated as $p_\theta(z|x)$, which is the true posterior distribution of the latent space. Computing the posterior $p_\theta(x)$ for each data sample x is expensive, so to limit the space for a faster search process, the variational autoencoder uses the approximate inference of the tractable posterior distribution, which is provided by the encoder function via $q_\varphi(z|x)$.

To ensure that both $p_\theta(z|x)$ and $q_\varphi(z|x)$ are similar, the divergence between them should be minimized. To determine the value of the distance between these two distributions, the difference measurement can be obtained with the help of the Kullback-Leibler divergence. The divergence, according to Eq. (2.27), is measured as KL divergence and always has a non-negative value:

$$D_{KL} = \left(q_\varphi(z|x) \parallel p_\theta(z|x)\right) \tag{2.27}$$

The loss function consists of two expressions, one related to reconstruction errors and the other related to KL divergence. The loss function is calculated using Eq. (2.28):

$$E_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\varphi(z|x) \parallel p_\theta(z)\right) = -\text{ELBO} \leq \log p_\theta(x) \tag{2.28}$$

Figure 2.17 shows the general architecture of the variational autoencoder.

2.4.2 Generative Adversarial Networks

The generative adversarial network (GAN) is a deep learning approach proposed by Goodfellow in 2014 [52]. GANs provide an alternative approach to techniques based on maximum likelihood estimation. GAN is an unsupervised deep learning approach where two neural networks compete in a zero-sum game. One of these networks is a

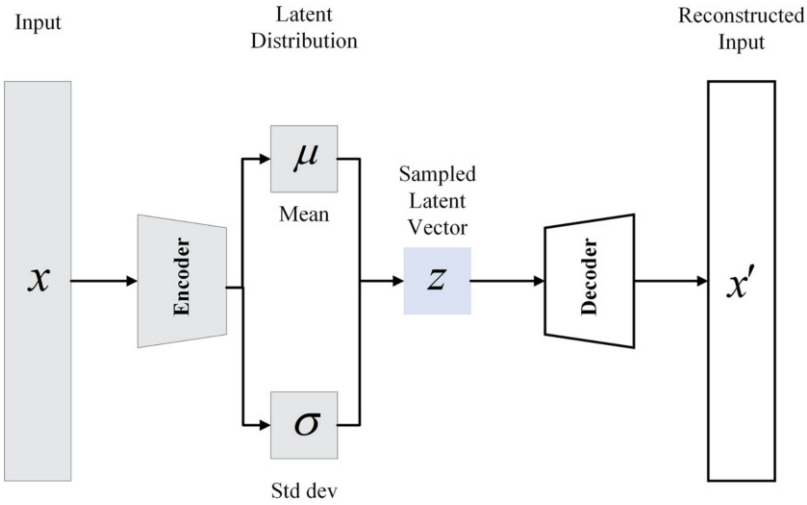

Fig. 2.17 The general architecture of variational autoencoders

generator, and the other is a discriminator. These two networks have an adversarial relation; each is defined as follows:

- **Generator:** A neural network that takes a random noise vector as input and transforms it into a distribution model.
- **Discriminator:** A neural network that distinguishes between output data (fake) and training data samples (real). It acts like a classifier and decides whether the input is real or fake. These two neural networks try to work against each other. The generator learns to transform a random noise vector into a distribution model at these weight settings.

For example, in image generation, the generator network starts generating images with Gaussian noise, and the discriminator network determines how good the generated images are. This process continues until the outputs of the generative network are close to the real input samples. Figure 2.18 presents the general architecture of GAN models.

The generator and discriminator network in GAN are two players playing the min-max game with the function *V (D, G)*, which can be expressed as Eq. (2.29):

$$\min_G \max_D V(D,G) = E_x[\log D(x)] + E_z[\log(1 - D(G(z)))] \tag{2.29}$$

In this regard, G and D represent the generator and the discriminator model, respectively, E_x is the expected value over all real data instances, and E_z is the expected value over all random inputs to the generator (the expected value over all generated fake instances $G(z)$). In practice, this equation may not provide adequate gradients to learn G (started from Gaussian random noise) in the initial stages. At the initial stages, D can reject samples because they are clearly different compared to the

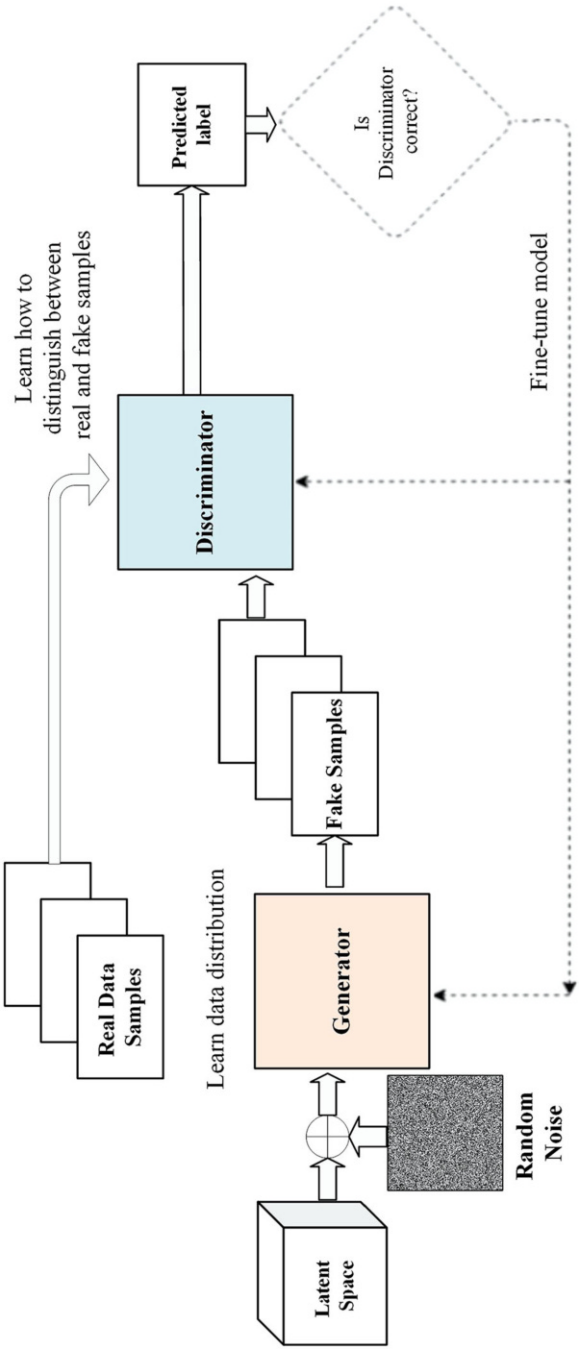

Fig. 2.18 The general architecture of GANs

training samples. In this case, the $\log(1 - D(G(z)))$ expression will be saturated. Instead of training G to minimize $\log(1 - D(G(z)))$,G can be trained to maximize $\log(G(z))$, which provides much better gradients at the initial stages of learning.

In general, the GAN network is designed for unsupervised learning tasks, but it has also proven to be a better solution for semi-supervised and reinforcement learning depending on the nature of the problem [53]. GANs are also used in transfer learning research to apply the alignment of the latent feature space. The application areas of GANs include healthcare, image analysis, data augmentation, video generation, audio generation, traffic control, cyber security, and many other evolving applications. Overall, GANs have established themselves as a comprehensive domain of data-independent expansion and a solution to problems requiring a generative approach.

2.4.3 Boltzmann Machines

Boltzmann machines (BMs) are undirected networks consisting of many nodes connected to each other through weighted connections [54]. BMs represent a class of unsupervised neural networks that do not attempt to minimize loss or achieve a goal; instead, they generate data to form a system (usually a probability distribution) similar to the original system [55]. Figure 2.19 shows the general architecture of BMs, where visible and hidden nodes are selected and visible nodes are used as input and output. Because, after feeding the visible nodes through contrastive divergence, which uses Gibbs' sampling, the visible nodes iteratively feed the hidden nodes through weights. Instead, hidden nodes feed visible nodes. In this figure, gray nodes are hidden, and white nodes are visible.

In this model, each node communicates with every other node, and the whole model works as a system to create a generative network. Therefore, it can make its own data based on what it learned by fitting it into a dataset. The visible nodes in Boltzmann machines can be interacted with, but not with the hidden ones. Another distinction is that there is no training process. The nodes learn how to model the dataset as best as possible, turning the Boltzmann machine into an unsupervised deep learning model.

If the units are updated sequentially in any order that does not depend on their total inputs, the network will finally reach a Boltzmann distribution. The probability of a state vector, v, is only determined by its "energy" relative to the energies of all possible binary state vectors u, which can be calculated using Eq. (2.30):

$$P(v) = e^{-E(v)} / \sum_{u} e^{-E(u)} \tag{2.30}$$

As in Hopfield nets, the energy of state vector v is defined as Eq. (2.31):

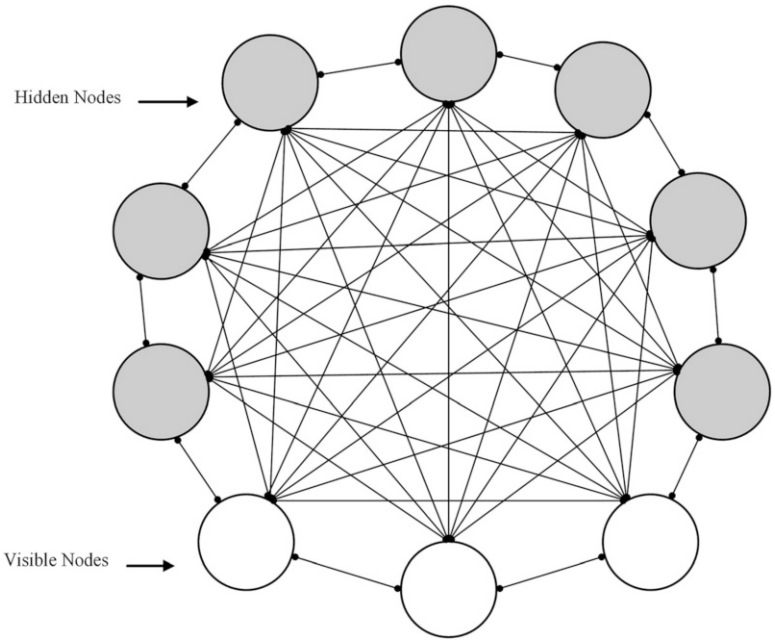

Fig. 2.19 The general architecture of Boltzmann machines

$$E(v) = \sum_i s_i^v b_i - \sum_{i<j} s_i^v s_j^v w_{ij} \tag{2.31}$$

where s_i^v is the binary state assigned to unit i by state vector v.

However, Boltzmann machines are not necessarily practical and suffer from problems when the network becomes larger. Therefore, different types of Boltzmann machines have been proposed, applicable in various domains, such as restricted Boltzmann machines (RBM) [56], deep Boltzmann machines (DBM) [57], and deep belief networks (DBN) [58], which are discussed and reviewed in the following subsections.

2.4.3.1 Restricted Boltzmann Machine

In practice, it is not easy to sample every iteration when all nodes are connected to every other node. Hence, the restricted Boltzmann machine was proposed [56]. RBM is similar to BM, but the main difference is that RBM consists of only two layers: the input layer and the hidden layer. Its architecture is similar to the artificial neural network model, so the RBM layers are like the first two layers of an

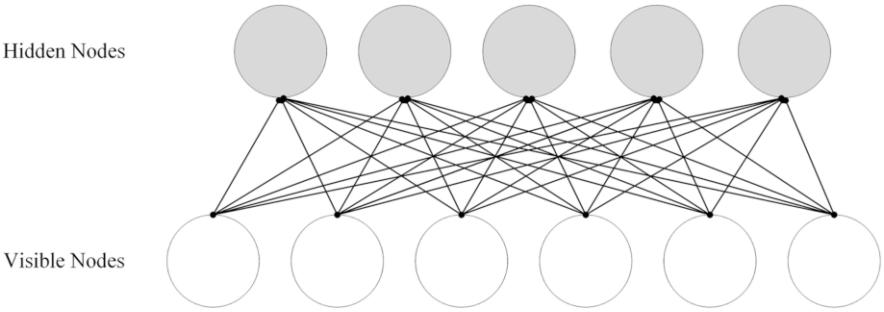

Fig. 2.20 The general architecture of RBMs

ANN. However, for the layers, there is a restriction that none of the nodes within the layer are connected to each other. RBM is Boolean/Bernoulli if each node outputs a binary value. Figure 2.20 shows the architecture of RBM.

RBMs (and BMs), in general, are energy-based models where the joint configuration energy of visible and hidden nodes *E(v, h)* is calculated based on Eq. (2.32):

$$E(v,h) = \sum_{i \in \text{visible}} p_i v_i - \sum_{j \in \text{hidden}} q_i h_i - \sum_i \sum_j v_i w_{i,j} h_j \tag{2.32}$$

In the above equation, v_i and h_i are the states of visible node i and hidden node j, p_i and q_i are their biases, and $w_{i,\ j}$ shows the weight between them. RBM uses Eq. (2.33) to determine the probability between each pair of hidden and visible vectors:

$$p(u,h) = \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \tag{2.33}$$

Using the contrastive divergence method, the lowest energy state is obtained by adjusting the weights. RBMs are used in the fields of dimensionality reduction, classification, regression, collaborative filtering, feature learning, topic modeling, etc. In general, RBMs can automatically recognize patterns in data and develop probabilistic or stochastic models, which are used to select or extract features.

2.4.3.2 Deep Belief Network

A deep belief network (DBN) is a multilayer generative graphical model that stacks and sequentially connects several individual unsupervised networks such as AE or RBM [58]. This type of network uses the hidden layer of each network as input for the next layer. One of the most important advantages of DBN, in contrast to conventional shallow learning networks, is the capability to detect deep patterns,

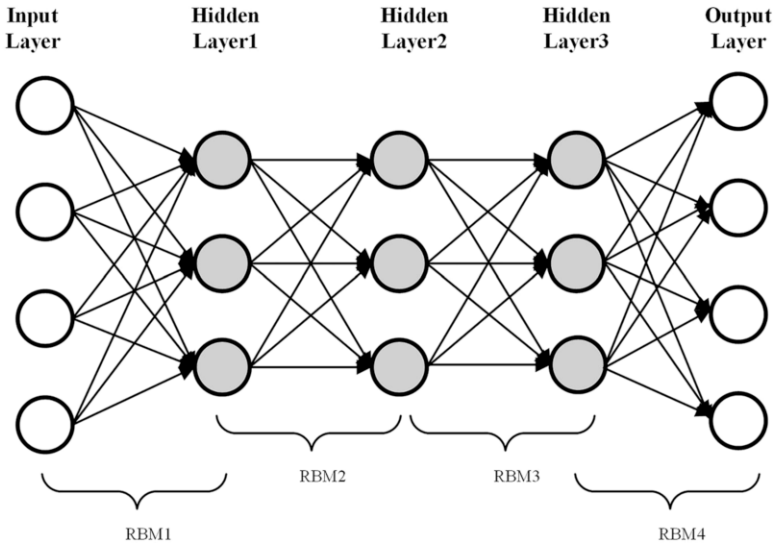

Fig. 2.21 The general architecture of DBNs

which allows to reason and identify the deep differences between normal and noisy data [59].

Considering that DBNs are extensions of RBMs, however, training DBNs is not simple because there is a phenomenon called "explaining away" in Bayesian networks, which happens when hidden variables are inferred in hidden layers. The reason for this is the intractability of the posterior distribution on the hidden variables. Explaining away happens when a cause of an effect completely explains the effect, which reduces the chance that other causes are involved. Markov chain Monte Carlo can be used to sample these intractable posterior distributions, but it is very time-consuming. Another problem with training DBNs is when the prior is assumed to be independent of the deepest hidden layer with initial random weights. To train DBNs faster and more efficiently, it is necessary to remove the "explaining away" effect and prior independence [55].

In deep belief networks, the training method is started with the first RBM, then the hidden layer plays the role of the visible layer of the second RBM, and the second RBM is trained. This process continues until each model layer is trained. Figure 2.21 shows the general architecture of a DBN with three hidden layers.

A continuous DBN is also simply an extension of a standard DBN that allows a continuous range of decimals instead of binary data. In general, the DBN model can play a key role in several high-dimensional data applications due to its effective feature extraction and classification capabilities and has become one of the important topics in the domain of neural networks [6].

2.4.3.3 Deep Boltzmann Machine

The RBM model can only learn a simple representation of the data. By increasing its hidden layers, it should be possible to build a deep architecture to extract the complex features of the input data. The deep Boltzmann machine (DBM) model is a deep generative model that has one visible layer and several hidden layers [57]. The general architecture of DBM is shown in Fig. 2.22.

The main difference between DBMs and DBNs is that in DBMs, all connections are undirected. DBMs are also used to capture hidden complex fundamental features in data, making them suitable for tasks such as speech and object recognition. DBMs, unlike DBNs, first use an approximate inference method with an additional bottom-up pass to speed up learning and incorporate top-down feedback, which makes the DBM deal well with ambiguous inputs.

The traditional process of training BMs uses a random start to approximate the gradients of the likelihood function for the input data, which is not the fastest approach. To deal with this, Salakhutdinov et al. proposed a variational technique using mean field inference to estimate the expectations associated with the data with a Markov chain-based estimation method to estimate the model's expected statistics [57]. The method involves Markov chains that initialize the weights to appropriate values to facilitate joint learning of all layers. However, it is costly compared to DBN pre-training, where inference is performed via a bottom-up pass. Therefore, the inference of DBMs is accelerated using detection weights.

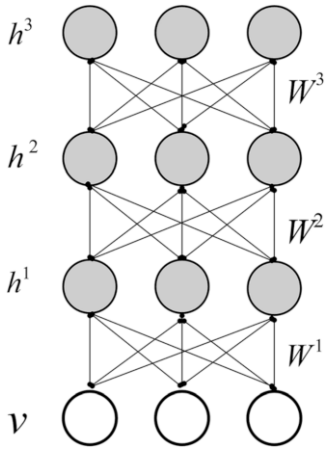

Fig. 2.22 The general architecture of DBMs

2.5 Graph-Based Models

Basic types of neural networks can only be implemented using regular or Euclidean data. Deep learning effectively detects hidden patterns in Euclidean data, while many data in the real world have a non-Euclidean graph structure. The number of problems in which data is represented in the form of graphs is increasing, such as graph/node classifications, community detection, link prediction, influencer identification, and much more.

Graphs are a type of data structure that models a set of objects (nodes) and their relations (edges). Recently, research on graph analysis with machine learning has received more and more attention due to the high representation power of graphs [60]. As a unique non-Euclidean data structure for machine learning, graph analysis focuses on different types of tasks. It should be mentioned that graph datasets are different from other datasets, such as text, image, audio, etc.

Graph-based models have the following unique characteristics:

- **Irregular domains:** It is possible to represent irregular domains or non-Euclidean data in graphs, while other datasets, such as images and audio, can be easily represented in a Euclidean plane or grid structure.
- **Non-static structure:** Graphs are a tool for representing complex systems. Therefore, they have different types, such as homogeneous, non-homogeneous, signed, unsigned, etc., and they may be node-oriented, graph-oriented, or edge-oriented. One of the most widely used methods of graph representation is the use of proximity. Matrices that change shape after nodes are added or removed. This is why conventional machine learning models cannot handle adjacency matrices directly.
- **Scalability and parallelization:** Big data is a problem in the era of an abundance of computing tools. As a result, the generated graphs may have millions of nodes and billions of edges. The second problem is how to parallelize the algorithms because each node in the graph contains information about other nodes in the graph, that is, nodes have relations with other nodes that should not be lost.
- **Domain-specific knowledge:** Graph learning may require specific knowledge of the domain that may help to make better predictions. Other additional information may also be useful to detect a new target or feature.

Graph neural networks (GNNs) are deep learning-based methods that operate on the graph domain. Early studies on GNNs and their concept were done by Scarselli et al. [61]. Based on the concepts of GNNs, many deep learning graph-based models such as graph convolutional networks (GCN) [62], graph attention networks (GAT) [63], and gated graph sequence neural networks (GGS-NN) [64] (a combination of GNN and a type of RNN) have been developed and used in various research.

GCNs are a class of neural networks that use convolution operations to extract meaningful statistical patterns from graph data and are capable of efficient implementation with minimal training [62]. In GAT, it is assumed that the influence of neighbors is not only not the same but also not pre-determined by the graph

structure, so it differentiates the contribution of neighbors using the attention mechanism. In GGS-NN, the recurrent function is executed several times on all nodes in the aggregation phase and GRUs are adapted for updating purposes. In addition to graph neural networks, GCNs have been used in session-based recommender system in various articles, which are discussed below in the following subsections.

2.5.1 Graph Neural Network

Graph neural networks (GNNs) are a set of methods that apply deep neural networks to graph-structured data. Classical deep neural networks cannot be easily extended to graph-structured data because the graph structure is not a regular network. The study of graph neural networks started in the early twenty-first century when the first GNN model was proposed for node- and graph-centric tasks [61]. A graph neural network is a type of neural network whose input data is a graph and learns to present the features of each node. Furthermore, generated features can be used to solve any graph-related problem, such as node classification, graph classification, and clustering. Graph neural networks can be considered as a process for learning a representation of data on a graph. GNNs focus on learning efficient features for each node to facilitate node-centric tasks. For graph-centric tasks, they learn features for the entire graph, where node feature learning is usually performed as an intermediate step. The node feature learning process usually uses both the input node features and the graph structure [65].

In the node classification problem, each node is characterized by its features x_i and is labeled l_v. In the graph classification problem, a set of nodes are associated with the label l_i. By learning the features of nodes i, the graph neural network can predict the labels of unknown nodes i. It learns to represent each node with a d-dimensional vector v_i. The vector v_i contains information about the neighbor nodes of node i, which is presented in Eq. (2.34) [66]:

$$V_i = f\left(X_i, X_{co[i]}, V_{ne[i]}, X_{ne[i]}\right) \tag{2.34}$$

In Eq. (2.34), $X_{co[i]}$ represents the features of edges adjacent to node i, and f is a transfer function (feedforward neural network) that outputs the d-dimensional vector. The above formula can be solved using the neighborhood aggregation theorem and iteratively rewritten as Eq. (2.35):

$$V^{t+1} = F(V^t, X) \tag{2.35}$$

The output transfer function O_i is applied to obtain the final vector with low dimensions as Eq. (2.36):

$$O_i = g(V_i, X_i) \tag{2.36}$$

Other hidden parameters are learned by applying the loss function between the predicted output O_i and the actual labels l_i.

2.5.2 Graph Convolutional Network

Graphs, which represent entities and their relations, are everywhere in the real world, such as social networks, traffic networks, knowledge graphs, molecular structures, etc. Recently, many studies have focused on developing deep learning approaches for graph data, leading to the rapid development in the domain of graph neural networks. Moreover, graph convolutions adopt a neighborhood integration (or message passing) scheme to learn representations of nodes by considering node features and graph topology information together, among which the most prominent method is graph convolutional networks (GCNs) [62]. Similar to convolutional neural networks that help speed up learning and increase accuracy with hierarchical data processing, graph convolutional networks are also meant to do the same thing but on graph data. Graph convolutional networks are a group of powerful neural networks that use convolution operations to extract meaningful statistical patterns from graph data and can perform efficiently with minimal training. In fact, they are so powerful that even a graph convolutional network with two randomly initialized layers can obtain a useful representation of node features. In general, graph convolutional networks find a new representation of each graph vertex with the integrity of the features of its neighbors.

Differentiating, pooling, and flattening are the functions that are utilized in convolutional graph networks. These three functions are important for the performance of this type of network and are common to all graph convolutional networks. A filter is a function that limits the number of cells to be considered at a time. The pooling function produces output for all values in a specified range at once, based on a function of maximum, average, etc. The flattening function transforms the network structure into a lower-dimensional vector whose outputs can be used as inputs to feedforward neural networks.

The node representation after a single layer of GCN can be defined as Eq. (2.37):

$$H = f\left(\left(\tilde{D}^{\frac{-1}{2}}(A+I)\tilde{D}^{\frac{-1}{2}}\right)XW\right) \tag{2.37}$$

In the above equation, $W \in \mathbb{R}^{d \times d}$ includes the network parameters, A is the node adjacency, $\tilde{D}_{ii} = \sum_j (A+I)_{ij}$, and f is an activation function. The representation of the node v after k layers can be written as Eq. (2.38):

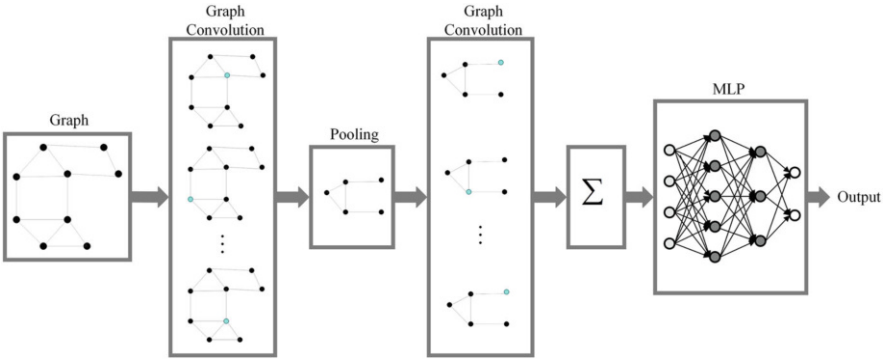

Fig. 2.23 The general architecture of GCNs

$$h_v = f\left(\sum_{u \in N(v)} \left(W^k h_u^k + b^k\right)\right), \quad \forall v \in V \tag{2.38}$$

where W^k and b^k represent the weight and bias parameters of GCN layer, respectively. $N(v)$ includes the nodes' neighborhood of v in graph G including v, and h_v is the representation of node v.

In the past decades, researchers have worked on how to perform convolutional operations on graphs. One approach is to define graph convolutions from a spectral point of view and another from a spatial point of view. Briefly, spectral graph convolutions are defined based on the Fourier transform of the graph, corresponding to the 1D Fourier transform. In this way, spectral-based graph convolutions can be computed by taking the inverse Fourier transform of the product between two Fourier transform signals. On the other hand, spatial graph convolutions can also be defined as the aggregation of node representations from neighbor nodes. This perspective is very effective for graph convolutional networks [67]. In general, graph convolutional network models are a type of neural network architecture that can convolutionally collect graph structure and node information from neighborhoods. Figure 2.23 shows the general architecture of graph convolutional networks.

2.6 Conclusion

Deep learning is a relatively new topic, defined as a set of layers that perform non-linear processing to learn different levels of data representation. For decades, researchers have been trying to discover patterns and data representations from raw data based on machine learning methods. Unlike conventional machine learning and data mining approaches, deep learning can generate very high-level data representations from huge amounts of raw data. Therefore, it is a solution for many real-

world applications. The evaluation results of deep learning methods and their comparison with the results of other methods show the capabilities of deep learning in different scopes.

In this chapter, first, an overview of the history and concepts of deep learning was presented, and the characteristics of deep learning and machine learning were compared. Then, a taxonomy of deep learning models was presented, in which they were divided into three general categories: discriminative, generative, and graph-based models. Finally, the fundamental models of these three categories were briefly discussed.

References

1. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016. https://www.deeplearningbook.org
2. Merima Kulin, Carolina Fortuna, Eli De Poorter, Dirk Deschrijver, and Ingrid Moerman. "Data-driven design of intelligent wireless networks: An overview and tutorial." Sensors 16, no. 6 (2016): 790. https://doi.org/10.3390/s16060790
3. Vasant Dhar. "Data science and prediction." Communications of the ACM 56, no. 12 (2013): 64-73. https://doi.org/10.1145/2500499
4. Paul Fergus, and Carl Chalmers. Applied Deep Learning: Tools, Techniques, and Implementation. Springer Nature, 2022. https://doi.org/10.1007/978-3-031-04420-5
5. Sergey I. Nikolenko. Introduction: The Data Problem. In: Synthetic Data for Deep Learning. Springer Optimization and Its Applications 174, (2021). https://doi.org/10.1007/978-3-030-75178-4_1
6. Iqbal H. Sarker. "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions." SN Computer Science 2, no. 6 (2021): 420. https://doi.org/10.1007/s42979-021-00815-1
7. Andrew Ng, and Michael Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." Advances in neural information processing systems 14 (2001). https://dl.acm.org/doi/abs/10.5555/2980539.2980648
8. Volodymyr Kuleshov, and Stefano Ermon. "Deep hybrid models: Bridging discriminative and generative approaches." In Proceedings of the Conference on Uncertainty in AI (UAI). Sydney, Australia, August 12-14, 2017.
9. Li Deng, and Navdeep Jaitly. "Deep discriminative and generative models for speech pattern recognition." In Handbook of pattern recognition and computer vision, pp. 27-52. 2016. https://doi.org/10.1142/9789814656535_0002
10. Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." IEEE transactions on pattern analysis and machine intelligence 35, no. 8 (2013): 1798-1828. https://doi.org/10.1109/TPAMI.2013.50
11. Léon Bottou. "Large-scale machine learning with stochastic gradient descent." In Proceedings of 19th International Conference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers, pp. 177-186. Physica-Verlag HD, 2010. https://doi.org/10.1007/978-3-7908-2604-3_16
12. Dong C. Liu, and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization." Mathematical programming 45, no. 1-3 (1989): 503-528. https://doi.org/10.1007/BF01589116
13. Diederik P. Kingma and Jimmy Lei Ba. "Adam: a Method for Stochastic Optimization". International Conference on Learning Representations, San Diego, CA, USA, May 7-9, 2015, page 1-13. https://arxiv.org/pdf/1412.6980.pdf

14. Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. "SGD: General analysis and improved rates." International conference on machine learning, Long Beach, CA, USA, Jun 10-15, 2019 pp. 5200-5209. PMLR, 2019. https://doi.org/10.48550/arXiv.1901.09401
15. John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of machine learning research 12, no. 7 (2011). https://jmlr.org/papers/v12/duchi11a.html
16. Matthew D. Zeiler. "Adadelta: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012). https://doi.org/10.48550/arXiv.1212.5701
17. Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark." Neurocomputing (2022). https://doi.org/10.1016/j.neucom.2022.06.111
18. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324. https://doi.org/10.1109/5.726791
19. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Communications of the ACM 60, no. 6 (2017): 84-90. https://doi.org/10.1145/3065386
20. Karen Simonyan, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014). https://doi.org/10.48550/arXiv.1409.1556
21. Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013). https://doi.org/10.48550/arXiv.1312.4400
22. Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and M. Riedmiller. "Striving for Simplicity: The All Convolutional Net." In ICLR (workshop track). 2015.
23. Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, USA, July 21-26, 2017 pp. 2261-2269. https://doi.org/10.1109/CVPR.2017.243
24. Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. "FractalNet: Ultra-Deep Neural Networks without Residuals." In International Conference on Learning Representations ICLR 2017, Toulon, France, April 24-26, 2017. https://doi.org/10.48550/arXiv.1605.07648
25. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, MA, USA, June 7-12, 2015, pp. 1-9. 2015.
26. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, Caesars Palace, Jun 26-July 1, 2016 pp. 770-778. https://doi.org/10.1109/CVPR.2016.90
27. Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions." Journal of big Data 8 (2021): 1-74. https://doi.org/10.1186/s40537-021-00444-8
28. Alex Sherstinsky. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network." Physica D: Nonlinear Phenomena 404 (2020): 132306. https://doi.org/10.1016/j.physd.2019.132306
29. Michael I. Jordan. "Serial order: A parallel distributed processing approach." In Advances in psychology, vol. 121, pp. 471-495. North-Holland, 1997. https://doi.org/10.1016/S0166-4115(97)80111-2
30. Jeffrey L. Elman. "Finding structure in time." Cognitive science 14, no. 2 (1990): 179-211. https://doi.org/10.1207/s15516709cog1402_1

31. G. R. Kanagachidambaresan, Adarsha Ruwali, Debrup Banerjee, and Kolla Bhanu Prakash. "Recurrent neural network." Programming with TensorFlow: Solution for Edge Computing Applications (2021): 53-61. https://doi.org/10.1007/978-3-030-57077-4_7
32. Sepp Hochreiter, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735
33. Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. "Sequence labelling in structured domains with hierarchical recurrent neural networks." In Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India. 6 – 12 January, 2007.
34. Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. "An application of recurrent neural networks to discriminative keyword spotting." In Artificial Neural Networks–ICANN 2007: 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part II 17, pp. 220-229. Springer Berlin Heidelberg, 2007. https://doi.org/10.1007/978-3-540-74695-9_23
35. Alex Graves, and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." Neural networks 18, no. 5-6 (2005): 602-610. https://doi.org/10.1016/j.neunet.2005.06.042
36. Tara N. Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. "Convolutional, long short-term memory, fully connected deep neural networks." In 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP), South Brisbane, Queensland, Australia, April 19-24, 2015, pp. 4580-4584. Ieee, 2015. https://doi.org/10.1109/ICASSP.2015.7178838
37. Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. "Multi-dimensional recurrent neural networks." In Artificial Neural Networks–ICANN 2007: 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part I 17, pp. 549-558. Springer Berlin Heidelberg, 2007. https://doi.org/10.1007/978-3-540-74690-4_56
38. Xiaodan Liang, Xiaohui Shen, Donglai Xiang, Jiashi Feng, Liang Lin, and Shuicheng Yan. "Semantic object parsing with local-global long short-term memory." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Caesars Palace, Jun 26-July 1, 2016, pp. 3185-3193. 2016.
39. Mike Schuster, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." IEEE transactions on Signal Processing 45, no. 11 (1997): 2673-2681. https://doi.org/10.1109/78.650093
40. Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. "A review of recurrent neural networks: LSTM cells and network architectures." Neural computation 31, no. 7 (2019): 1235-1270. https://doi.org/10.1162/neco_a_01199
41. Kyunghyun Cho, Bart Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, Doha, Qatar, October 25-29, 2014, pp.1724-1734. 2014. https://doi.org/10.3115/v1/D14-1179
42. Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In 3rd International Conference on Learning Representations, ICLR 2015. 2015.
43. Jakub M. Tomczak, "Deep Generative Modeling", Springer Nature, 2022, https://doi.org/10.1007/978-3-030-93158-2
44. Lars Ruthotto, and Eldad Haber. "An introduction to deep generative modeling." GAMM-Mitteilungen 44, no. 2 (2021): e202100008. https://doi.org/10.1002/gamm.202100008
45. Mark A. Kramer. "Nonlinear principal component analysis using autoassociative neural networks." AIChE journal 37, no. 2 (1991): 233-243. https://doi.org/10.1002/aic.690370209
46. R. Indrakumari, T. Poongodi, and Kiran Singh. "Introduction to Deep Learning." Advanced Deep Learning for Engineers and Scientists: A Practical Approach (2021): 1-22. https://doi.org/10.1007/978-3-030-66519-7_1
47. Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. "Contractive auto-encoders: Explicit invariance during feature extraction." In Proceedings of the 28th

international conference on international conference on machine learning, Bellevue, USA, June 28 - July 2, 2011, pp. 833-840.
48. Alireza Makhzani, Brendan Frey. K-sparse autoencoders. 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014.
49. Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of machine learning research 11, no. 12 (2010).
50. Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. "Stacked convolutional auto-encoders for hierarchical feature extraction." In Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21, pp. 52-59. Springer Berlin Heidelberg, 2011. https://doi.org/10.1007/978-3-642-21735-7_7
51. DP Kingma, Welling M. Auto-encoding variational bayes. 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014.
52. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets" (Advances in neural information processing systems) (pp. 2672–2680). Red Hook, NY Curran (2014).
53. Alankrita Aggarwal, Mamta Mittal, and Gopi Battineni. "Generative adversarial network: An overview of theory and applications." International Journal of Information Management Data Insights 1, no. 1 (2021): 100004. https://doi.org/10.1016/j.jjimei.2020.100004
54. David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. "A learning algorithm for Boltzmann machines." Cognitive science 9, no. 1 (1985): 147-169. https://doi.org/10.1016/S0364-0213(85)80012-4
55. Harshvardhan GM, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. "A comprehensive survey and analysis of generative models in machine learning." Computer Science Review 38 (2020): 100285. https://doi.org/10.1016/j.cosrev.2020.100285
56. Geoffrey E. Hinton. "A practical guide to training restricted Boltzmann machines." Neural Networks: Tricks of the Trade: Second Edition (2012): 599-619. https://doi.org/10.1007/978-3-642-35289-8_32
57. Ruslan Salakhutdinov, and Geoffrey E. Hinton. Deep Boltzmann machines, in: Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, Florida, USA, April 2009.
58. Geoffrey E. Hinton. "Deep belief networks." Scholarpedia 4, no. 5 (2009): 5947. https://doi.org/10.4249/scholarpedia.5947
59. Jing Ren, Mark Green, and Xishi Huang. "From traditional to deep learning: Fault diagnosis for autonomous vehicles." In Learning Control, pp. 205-219. Elsevier, 2021. https://doi.org/10.1016/B978-0-12-822314-7.00013-4
60. Ziwei Zhang, Peng Cui, and Wenwu Zhu. "Deep learning on graphs: A survey." IEEE Transactions on Knowledge and Data Engineering 34, no. 1 (2020): 249-270. https://doi.org/10.1109/TKDE.2020.2981333
61. Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model." IEEE transactions on neural networks 20, no. 1 (2008). https://doi.org/61-80.10.1109/TNN.2008.2005605
62. Thomas N. Kipf, and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In International Conference on Learning Representations ICLR 2017, Toulon, France, April 24-26, 2017.
63. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018.
64. Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. "Gated Graph Sequence Neural Networks." 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016.

65. Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. "Graph neural networks: A review of methods and applications." AI open 1 (2020): 57-81. https://doi.org/10.1016/j.aiopen.2021.01.001
66. Shi Dong, Ping Wang, and Khushnood Abbas. "A survey on deep learning and its applications." Computer Science Review 40 (2021): 100379. https://doi.org/10.1016/j.cosrev.2021.100379
67. Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. "Graph convolutional networks: a comprehensive review." Computational Social Networks 6, no. 1 (2019): 1-23. https://doi.org/10.1186/s40649-019-0069-y

Chapter 3
Deep Discriminative Session-Based Recommender System

Abstract Due to the sequential nature and time-ordered session data, much research in a session-based recommender system (SBRS) focuses on recurrent neural networks (RNNs), including GRU and LSTM. On the other hand, convolutional neural networks (CNNs) provide very effective solutions for modeling sequential data when sequence elements are associated with complex features. As a result, we discuss different deep discriminative models in SBRS in this chapter, such as variants of RNNs and CNNs.

Keywords Session-based recommender systems · SBRS · Deep discriminative models · RNN · LSTM · GRU · CNN

3.1 Introduction

In recent years, there has been increasing research progress in a session-based recommender system (SBRS), where almost all the proposed approaches utilized deep neural network architectures. Deep neural networks have provided excellent performance in some areas, such as image and speech recognition [1, 2], where unstructured data are processed through several convolutional and standard layers. The most popular of these networks are convolutional neural networks (CNNs). However, sequential data modeling based on different recurrent neural network (RNN) models has recently attracted much attention because it is the most appropriate model for analyzing this type of data. The use of deep neural networks in SBRS has significantly increased the accuracy and effectiveness of recommendations by overcoming the obstacles of conventional models and achieving high recommendation quality [3, 4]. Deep discriminative models, such as RNNs and CNNs, can effectively capture and analyze non-linear and non-obvious user/item relationships. Additionally, it can extract and aggregate complex relationships within data from different sources, such as textual and visual information.

As previously discussed in Chap. 1, the session-based recommender systems have eliminated the need to access interactions and the long-term interests of users. Indeed, they focus on the recent interactions of users and their short-term interests while considering the changes in these interests in short periods. However, the issue

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2_3

of scalability and managing the volume and variety of items and users is one of their major challenges. One of the important advantages of deep neural networks for session-based recommender systems is that they can adjust model parameters with different sizes of training datasets [3, 5]. In the learning process, changing and updating the hidden variables related to items and users are performed independently of previous data [6].

A review of published research shows that discriminative deep learning approaches in session-based recommender systems using GRU, LSTM, and CNN have received much attention. To this end, different approaches to deep discriminative models in SBRS are discussed and analyzed in this chapter, considering the models, datasets, evaluations, and highlights/limitations of each. First, in Sect. 3.2, a brief overview of the fundamentals related to this chapter of the book is provided, including the research distribution statistics, the employed datasets, and the evaluation methods/metrics used in various research. Then, in the next Sects. 3.3 and 3.4, approaches based on RNN and CNN are briefly discussed. Section 3.5 analyzes the results and identifies existing challenges related to the deep discriminative models in SBRS and provides several guidelines for future research.

3.2 Fundamentals

Before the emergence of deep learning techniques, limited research had been conducted in the field of session-based recommender systems because the complexity caused by the consecutive nature of the session data was difficult to analyze and model by common methods. The only technique that could be used in the session-based recommendation field was the item-to-item recommendation, in which the recommended items were selected based on the similarity between items of the user's previous events, and the rest of the session was ignored [7]. This method had very low accuracy because only items were recommended to the user that was similar to the previous items or had occurred in the past at the same time as the user's previously selected item. By presenting deep learning techniques, the ability to model the sequence of user interactions in one session or several other sessions was provided.

In recent years, different types of deep learning techniques have been used in session-based recommender systems, which, from a general perspective, can be classified into three categories: deep discriminative models, deep generative models, and hybrid/advanced models. Deep discriminative models include neural networks trained in a way that allows their output to be interpreted as approximate posterior class probabilities and directly compute the probability of an output given an input [8]. Recurrent neural networks (RNNs) and convolutional neural networks (CNNs) are types of these methods, each including various architectures. Generative and hybrid/advanced models in SBRS are discussed in the next chapters of the book.

Since 2016, different research on session-based recommender systems using deep learning has been published. The beginning of this path belongs to research

Table 3.1 The list of research discussed using deep discriminative models

Deep discriminative models	References
GRU	[9–29]
LSTM	[30–39]
CNN	[40–48]

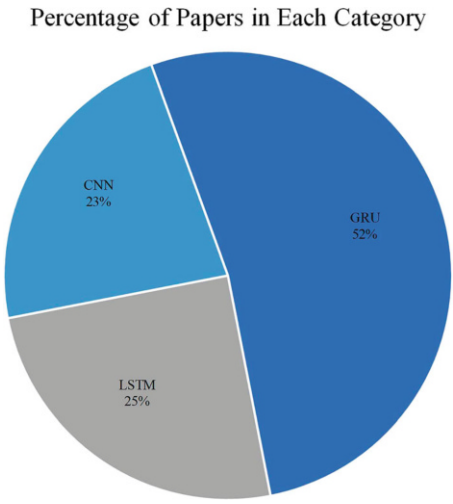

Fig. 3.1 Percentage of each type of deep discriminative model in SBRS

performed by Hidasi et al. that used the RNN technique in SBRS [9]. After that, more research was conducted using deep learning techniques in SBRS [10]. A review of various research related to the deep discriminative models in SBRS shows that RNNs and CNNs have been used in many articles. RNNs include GRU (gated recurrent unit), LSTM (long short-term memory), and the improved types of these two networks. CNNs classify into two-dimensional, three-dimensional, and dilated CNNs.

Table 3.1 summarizes the list of research reviewed in this chapter, separated based on the deep discriminative model.

To get a more comprehensive perspective of the research reviewed in this chapter, Fig. 3.1 shows the percentage of each technique used in the discussed research based on GRU, LSTM, and CNN.

According to Fig. 3.1, most of the reviewed research are based on two popular types of RNN, i.e., GRU and LSTM. Due to their sequential nature, RNNs have a great capacity to analyze the sequential dependencies between data in user sessions. Indeed, the ability to model the dynamic behavior of users over time in session-based recommender systems has made RNNs an appropriate solution in this scope. The GRU deep neural network has received more attention than LSTM, considering a large number of gates and parameters of the LSTM, which leads to higher computational complexity.

Figure 3.2 shows the general architecture of session-based recommender systems using RNNs. The user interactions, clicks, and other session data are provided to the

model as input and then converted into analyzable data structures using an embedding technique. Afterward, a type of RNN is utilized to model the structured data and discover their dependency relationships. Finally, and before the output layer, the fully connected layer is used to increase the stability of the model.

Some approaches related to deep session-based recommender systems use the CNN model. The use of CNN is suitable for user session data in two ways: (1) The sequence of items in one session or between different sessions of users can be easily implemented and modeled on CNN. (2) CNNs have a high capacity to learn the local and spatial features of regions and capture the related dependencies that are usually ignored by other models.

To learn and model the data related to users and items, these data should be embedded suitably in the CNN type of session-based recommender system, so that by successively executing convolution and pooling operations, temporal and spatial patterns between them are correctly identified. The user's favorable items are predicted based on the features captured from the input data and the dependencies between them.

Figure 3.3 shows the general architecture of session-based recommender systems using CNN, which can be used (or customized) to model data using various types of CNNs.

The following two subsections present a review and discussions of the dataset and evaluation methods used in the literature regarding deep discriminative approaches.

3.2.1 Datasets

Several well-known datasets have been employed for the evaluation purposes of session-based recommender systems using deep learning; each includes data related to different session features, such as events (interactions), items, and users. In fact, the authors usually select the most appropriate datasets considering the requirements of the proposed methodology and use them for assessment and comparison with other research.

Table 3.2 shows the datasets used in different articles, including the dataset name, the domain, a brief description, and the paper that employed it.

Characteristics of the most popular datasets, including the number of sessions, events, users, items, and data collection period, are listed in Table 3.3.

It is necessary to mention some points in Table 3.3. In YooChoose, Diginetica, Tmall, and RecSys Challenge 2015 datasets, sessions whose length is equal to 1 are not considered [7]. In the VIDEO dataset, sessions with a length of less than 3, users with less than 5 sessions, and items that have been repeated less than 10 times have been eliminated [14]. Because there are very few interactions for many items in the MovieLens dataset, in the preprocessing step, the items repeated less than 20 times were removed. Additionally, based on the timestamp of the interactions, the interactions related to a specific user are collected as a sequence and divided into k subsequences (watching k movies). Here, k is set to 30 and 100 for both datasets,

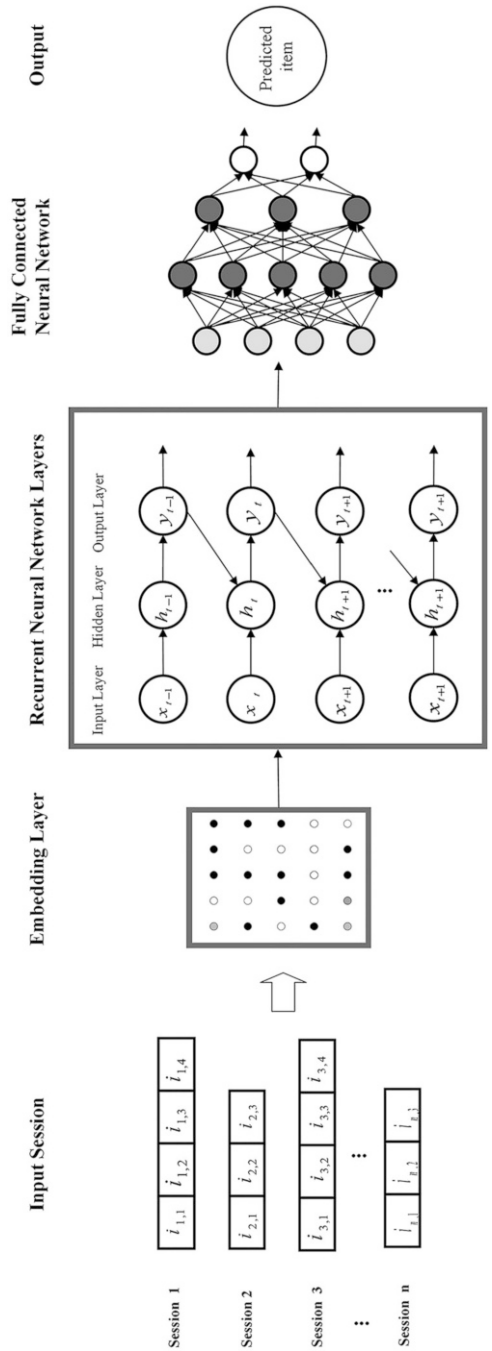

Fig. 3.2 The general architecture of session-based recommender systems using RNNs

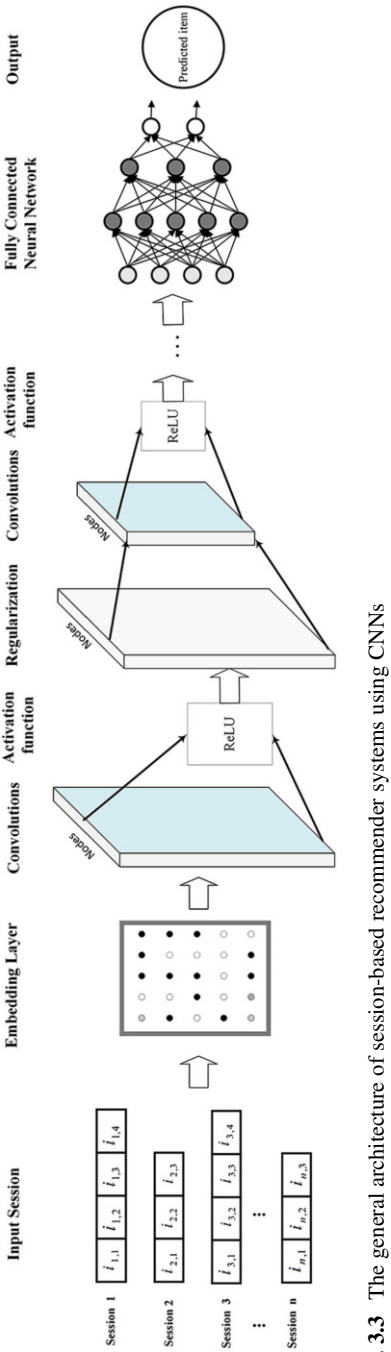

Fig. 3.3 The general architecture of session-based recommender systems using CNNs

Table 3.2 Widely used datasets in SBRS using deep discriminative models

Dataset	Domain	Description	References
Diginetica	E-commerce	The dataset includes user sessions extracted from an e-commerce search engine log	[20, 22, 26, 29, 45, 46]
YooChoose	E-commerce	The dataset consists of 6 months of clickstreams from an e-commerce Web site	[11, 20, 22, 26, 29, 33, 41, 45, 46, 48]
Gowalla	POI	This dataset is from a location-based social networking Web site where users share their locations by checking in	[22, 42, 47]
Last.fm	Music	This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system	[15, 16, 20, 22, 26, 27, 30, 41, 48]
RecSys Challenge 2015	E-commerce	This dataset comprises clickstream data about user sessions with an e-commerce Web site	[9, 13, 16, 17, 18, 21, 25, 28]
VIDEO	Video	This dataset is collected from a YouTube-like OTT video service platform and includes events of watching a video	[9, 14, 18, 21]
vidaXL	Video	This dataset is collected over a 2-month period from a YouTube-like video site and contains video-watching events	[10, 18]
CLASS	E-commerce	This dataset consists of product view events of an online classified site	[10, 18]
Internal	E-commerce	It contains user browsing and purchasing activity on multiple e-commerce Web sites from diverse verticals over the period of 3 months	[11]
XING	Job posting	It is the XING RecSys Challenge 2016 dataset that contains interactions on job postings. User interactions come with timestamps and interaction types (click, bookmark, reply, and delete)	[14]
Reddit	News	It is on user activity on the social news aggregation and discussion Web site Reddit	[15, 27]
Tmall	E-commerce	This is the large dataset released in the IJCAI-15 challenge, which has been collected from Tmall, the largest business-to-consumer e-commerce Web site in China. It records two types of user behaviors, views, and purchases	[16, 24, 42, 47]
AOTM	Music	This dataset includes the user-contributed playlists from the Art of the Mix Web site	[16]
8T	Music	This dataset includes the user-contributed playlists from the 8tracks.com (8T) Web site	[16]
MovieLens	Movie	It consists of users' sequential rating records for different categories of movies on the MovieLens site	[19, 23, 36, 42, 44]

(continued)

Table 3.2 (continued)

Dataset	Domain	Description	References
DoubanEvent	Movie	It is a Chinese Web site that allows Internet users to share their comments and viewpoints about movies in the Douban Movie Web site	[19]
Adressa	News	Adressa is a news dataset that contains reading behaviors and sessions from users	[23]
CiteULike	Research paper	In the CiteULike dataset, one user annotating one research paper at a certain time may have several records in order to distinguish different tags	[30]
Advertising dataset	Advertising	It is a public dataset released by Alimama, an online advertising platform in China. It contains records from ad display/clicks logs of users and ads in 8 days	[32]
Recommender dataset	E-commerce	This dataset contains many display/clicks logs of users and items on the Alibaba Web site	[32]
GHTorrent	GitHub	GHTorrent monitors the public event timeline of GitHub and provides abundant social relations between developers and development interactions between developers and software repositories from the popular collaborative social coding platform GitHub	[35]
Libraries.io	Software packages	Libraries.io provides explicit dependency relations between software packages	[35]
JEWELRY	E-commerce	It consists of product view events of a Web site selling jewelry products. The view events immediately followed by add-to-cart events were specially marked	[40]
ELECTRONICS	E-commerce	It consists of product viewing clicks of a Web site selling electronic products. The view events with the following add-to-cart were marked	[40]
Foursquare	POI	This dataset contains check-ins in NYC and Tokyo collected for about 10 months. Each check-in is associated with its timestamp, its GPS coordinates, and its semantic meaning	[42]
TW10	Video	TW10 is a short video dataset in which the average playing time of each video is less than 30 s	[44]
Retailrocket	E-commerce	The data has been collected from a real-world e-commerce Web site. It is raw data, i.e., without any content preprocessing; however, all values are hashed due to confidential issues	[45]

(continued)

Table 3.2 (continued)

Dataset	Domain	Description	References
Ta-Feng	E-commerce	The Ta-Feng dataset contains numerous baskets of purchased items from a grocery store, where each basket encapsulates the items purchased by one user in a period of time	[24]

respectively, resulting in ML30 and ML100 datasets. In each dataset, if the number of interactions is less than 10 and less than 20, the sequence will be deleted [44]. The Adressa dataset has two different versions, one of which was collected over 1 week and the other over 3 months [51]. The information presented in Table 3.3 is related to the preprocessed data of the second version, which is related to 16 days, and the sessions that include an interaction have been removed. Also, repeated clicks in each session have been removed [49]. In the Reddit dataset, if two clicks are more than 1 h apart, they are placed in two separate sessions [15].

3.2.2 Evaluation

Generally, two evaluation methodologies are used in session-based recommender system: online and offline. In the offline approach, parts of the user's feedback are unknown to the recommender system, but in the online, the system presents its recommendations to the actual user and receives her feedback. In each method, various evaluation metrics are used to evaluate the session-based recommender system so that the efficiency of the proposed method can be quantitatively measured and compared with others. According to the proposed method and the employed datasets, one or more well-known baselines are selected to compare the results.

The input data of session-based recommender systems usually are a sequence of user interactions, which in the offline evaluation approach are typically divided into two sets, train and test. Because the long-term history of the user's interactions is inaccessible in this type of system, they chose the last N sessions for testing purposes and split the data based on that. In online methods, users' feedback should be collected, which is usually either user answers to qualitative questions or field tests based on a production-like environment and collects implicit feedback from a large number of users [3]. Usually, in academic research, the possibilities of conducting large-scale online studies are often limited, and studies related to session-based recommender system mostly use offline evaluation methods [3].

For an accurate evaluation and comparison of the results of the proposed approaches, some previous methods are typically used, called as the baselines. Some of these methods have been developed using traditional session-based

Table 3.3 Characteristics of widely used datasets

Dataset	Number of sessions	Number of items	Number of events	Timespan	Average session length	Interaction type	Access link
YooChoose [7]	7,981,581	37,486	31,708,505	182 days	3.97	Click/buy	https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015
Diginetica [7]	204,789	43,136	993,483	–	4.85	Click	https://competitions.codalab.org/competitions/11161#learn_the_details-data2
RecSys Challenge 2015 [3]	7,981,581	37,486	31,708,505	182 days	3.97	Click/buy	https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015
Tmall [7]	1.77M	425,348	13.42M	91 days	7.56	Click/buy	https://ijcai-15.org/index.php/repeat-buyers-prediction-competition
VIDEO [14]	133,165	19,218	825,449	60 days	6.2	Click	Not public
MovieLens ML30 [44]	858,160	18,273	25,368,155	17 years	–	Rating (1–5)	http://files.grouplens.org/datasets/movielens/
MovieLens ML100 [44]	300,624	18,226	25,240,741	17 years	–	Rating (1–5)	http://files.grouplens.org/datasets/movielens/
Adressa [49]	982,210	13,820	2,648,999	16 days	2.7	Click	http://reclab.idi.ntnu.no/dataset
Reddit [15]	1,135,488	27,452	3,848,330	30 days	3	Click	https://www.kaggle.com/colemaclean/subreddit-interactions
Gowalla [22]	830,893	29,510	1,122,788	240 days	3.85	Check-in	https://snap.stanford.edu/data/loc-gowalla.html
Last.fm [50]	169,576	449,037	2,887,349	95 days	17.03	Click	http://millionsongdataset.com/lastfm/
AOTM [3]	21,888	91,166	306,830	95 days	14.02	Click	http://www.artofthemix.org
8T [3]	132,453	376,422	1.50 M	95 days	11.32	Click	Not public

recommender system techniques, and others utilize deep neural networks. The most frequently used baselines are as follows:

POP More popular items are always recommended. The POP is effective and straightforward simultaneously and is often a strong baseline in specific domains.

S-POP The most popular items in the current session are recommended. The recommendation list changes based on the number of events that are related to particular items. This baseline is useful for the domains with high repetitiveness.

Item-KNN Items similar to the actual item are recommended, and the similarity between them is measured based on the cosine similarity measure of their session vectors. In other words, it is the number of co-occurrences of two items in sessions divided by the square root of the product of the number of sessions in which the individual items occurred. This method is very effective for evaluating item-to-item recommendation methods [52].

BPR-MF It utilizes matrix factorization which is optimized for pairwise ranking objective functions through stochastic gradient descent. Methods based on matrix factorization cannot be used in session-based models because there is no pre-computed feature vector for new sessions. This problem is overcome by using the average vectors of the items that belong to each session [54].

FPMC A hybrid model for the next-basket recommendation based on the factorizing personalized Markov chains [53].

GRU4Rec A technique based on recurrent neural networks, which is one of the first approaches to using deep learning techniques in session-based recommender system. This method is based on GRU and is used to overcome the problem of gradient vanishing [9].

GRU4Rec+ This method is one of the most recent methods that extend GRU4Rec by introducing an improved sampling strategy pattern [25].

NARM An improved method of GRU4Rec, which improves session modeling by introducing a hybrid encoder based on the attention mechanism. In this technique, global and local encoders are defined, the global encoder corresponds to the GRU4Rec method, and the local encoder is proposed for adding the attention mechanism to the model, respectively [55].

STAMP This method is based on a Short-Term Attention/Memory Priority Model and, unlike the NARM method, is not based on a recurrent neural network. In this method, users' general interests are obtained through the long-term memory data of the session context, and their short-term interests are also recognized through short-term memory [56].

It should be noted that traditional evaluation methods were done by predicting the user's score for each item. Today, instead of using these methods, a list with a limited size, for example, 10 or 20, is considered for each user, which indicates the number of items at the top of the recommendation list. The quality of the

recommendation list in the test set is measured numerically by checking the number and rank of related items in the list. Some of the evaluation metrics that are used for this purpose have been listed below:

- *Recall:* This metric is calculated based on the number of relevant items that are among the top N items in the recommendation list, and the rank of the relevant items in the N list is unimportant, and it is calculated using Eq. (3.1):

$$\text{Recall@}N = \frac{\text{Number of relevant items in top } N \text{ list}}{\text{Total of relevant items}} \tag{3.1}$$

- *Mean Reciprocal Rank (MRR):* MRR focuses on the rank of relevant items in the list of recommendations. It shows that placing a relevant item at the top of the recommendation list significantly impacts user satisfaction and is calculated using Eq. (3.2):

$$\text{MRR@}N = \frac{1}{Q}\sum_{i=1}^{Q}\begin{cases}\frac{1}{\text{rank}_i} & \text{if } \text{rank}_i \le N \\ 0 & \text{otherwise}\end{cases} \tag{3.2}$$

  where Q is a sample of recommendation lists and rank_irefers to the rank position of the relevant item for the i-th recommendation list.

- *Precision @ N:* This metric evaluates the number of relevant items relative to the total N items recommended in the list, and it is calculated using Eq. (3.3):

$$\text{Precision@}N = \frac{\text{Number of relevant items in top } N \text{ list}}{\text{Total of } N \text{ items}} \tag{3.3}$$

- *Coverage@N:* It checks the coverage of the items. Item coverage measures the percentage of items that are ever recommended, and the variety of the recommended items in the recommendation list is considered. Its goal is to recommend a high percentage of various items to the user. This measure is calculated using Eq. (3.4):

$$\text{Coverage@}N = \frac{\text{distinct items that appeared in any top} - N \text{ recommendation}}{\text{all distinct recommendable items}} \tag{3.4}$$

- *Hit Rate@N:* It is the percentage of times in which relevant items are retrieved among the top N ranked items, and it is calculated using Eq. (3.5):

$$\text{Hit Rate@}N = \frac{1}{Q}\sum_{i=1}^{Q}\begin{cases} 1 & \text{if } \text{rank}_i \le N \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where Q is a sample of recommendation lists and rank_irefers to the rank position of the relevant item for the i-th recommendation list.

- *F1:* This metric is calculated based on a combination of precision and recall, and it is calculated using Eq. (3.6):

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.6}$$

- *$nDCG_p$:* This measure is based on cumulative gain (CG). The cumulative gain is the sum of the graded relevance values of all items in a recommendation list. nDCG is computed as the ratio between discounted cumulative gain (DCG) and idealized discounted cumulative gain (IDCG). Eqs. (3.7), (3.8), and (3.9) show how to calculate this measure:

$$\text{DCG}p = \sum_{i=1}^{p}\left(\frac{2^{r_i}-1}{\log_2(i+1)}\right) \tag{3.7}$$

$$\text{IDCG}p = \sum_{i=1}^{\text{REL}p}\left(\frac{r_i}{\log_2(i+1)}\right) \tag{3.8}$$

$$n\text{DCG}p = \frac{\text{DCG}p}{\text{IDCG}p} \tag{3.9}$$

In the above equations, r_i is the graded relevance of the result at position i, and RELp represents the list of relevant items (ordered by their relevance) up to position p.

- *MAP:* This metric calculates the average precision. In fact, after each relevant item is recommended, the precision is measured, and the average is calculated using Eq. (3.10):

$$\text{MAP} = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \tag{3.10}$$

In this relation, *P(q)* is the precision of query *q,* and parameter Q is the number of queries.

- *Mean Absolute Error (MAE):* This metric is one of the most common errors of prediction factors, which calculates the mean absolute value of the difference

between the score predicted by the system and the actual score of the item. The mean absolute error indicates the degree of closeness of the recommendations to reality, and it is calculated using Eq. (3.11):

$$\mathrm{MAE} = \left(\frac{1}{N} \sum_{i \in O_u} \left| P_{u,i} - r_{u,i} \right| \right) \tag{3.11}$$

- *Root Mean Square Error (RMSE):* The metric of the root mean square error of the predicted rank is more effective than the mean absolute error in problems where the errors are more considerable, and it is calculated using Eq. (3.12):

$$\mathrm{RMSE} = \sqrt{\frac{1}{N} \sum_{i \in O_u} \left(P_{u,i} - r_{u,i} \right)^2} \tag{3.12}$$

In Eqs. (3.11) and (3.12), $P_{u,i}$ is the predicted score for the item i by user u, $r_{u,i}$ is the actual value of the score assigned to item i by user u, O_u is the set of items rated by user u, and N is the total number of predictions made by the system.

- *Area Under the ROC Curve (AUC):* Another important metric used to determine the efficiency of recommender systems is the AUC. The larger the AUC value, the more favorable the final system performance is evaluated. The ROC (receiver operating characteristic) space is formed by two indices FPR on the horizontal axis and TPR on the vertical axis, as calculated by Eqs. (3.13) and (3.14), respectively. The line that connects two points (0,0) and (1,1) divides the ROC space into two parts. The area above this line is the favorable area and below the line is the unfavorable area. Therefore, the AUC is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.13}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{3.14}$$

The maximum value of this measure is equivalent to one and occurs in a situation where the recommender system is ideal and can recognize all positive samples. The AUC measure, unlike other measures for deciding the efficiency of classification methods, is independent of the classification threshold. Therefore, this metric indicates the output reliability of the system.

Table 3.4 shows the different evaluation metrics used in different articles on session-based recommender systems using deep discriminative models.

Table 3.4 Widely used evaluation metrics in SBRS using deep discriminative models

Evaluation metrics	References
Mean reciprocal rank (MRR)	[9, 10, 12–23, 25–29, 30, 31, 34, 40, 41, 43–45, 48]
Recall@n	[9–11, 13–15, 17–21, 23, 25, 27, 28, 30, 38, 40, 42, 57]
nDCG	[12, 24, 35, 41, 44, 48, 57]
AP	[42]
AUC	[12, 39]
Precision@n	[14, 22, 26, 29, 38, 39, 42, 45]
Hit Rate@n	[16, 35, 41, 44, 48]
F1	[24, 34]
Accuracy	[43]
RMSE	[43]
MAE	[27]
MAP	[39]

3.3 Session-Based Recommender System Using RNN

Before looking at the approaches of recurrent neural network models in session-based recommender systems, an overview of RNN, its variants, and the reasons that made it an effective choice for SBRS are provided.

3.3.1 Why RNN?

Sequential methods directly model the sequence of user actions instead of relying on features or co-occurrence frequencies. To be specific, recurrent neural networks (RNNs) are a class of deep neural networks that have been successfully used to predict the next item [9]. RNNs have a hidden state with non-linear dynamics that enables them to discover patterns of events, which lead to the prediction of the next item. In addition to the sequence of the items, more information about the user-item interactions is also available, such as the type of interaction, the time interval between events, and the time of interaction. This contextual information can significantly improve the prediction of the next event/item. For example, knowing the event type of past products or different patterns of time gaps between past user events can change the probability of the identity of the next product the user will interact with. Figure 3.4 shows the different patterns of time gaps in the next item prediction.

Leading research in this field is the GRU4Rec, which is based on RNN [9]. In this method, RNN is trained based on the features of a session, such as clicks related to item IDs, considering the ranking loss measure. However, GRU4Rec only focuses on clicked items in the current session, while extended models can also incorporate other user behavior during sessions (such as the length of time the user spends in a

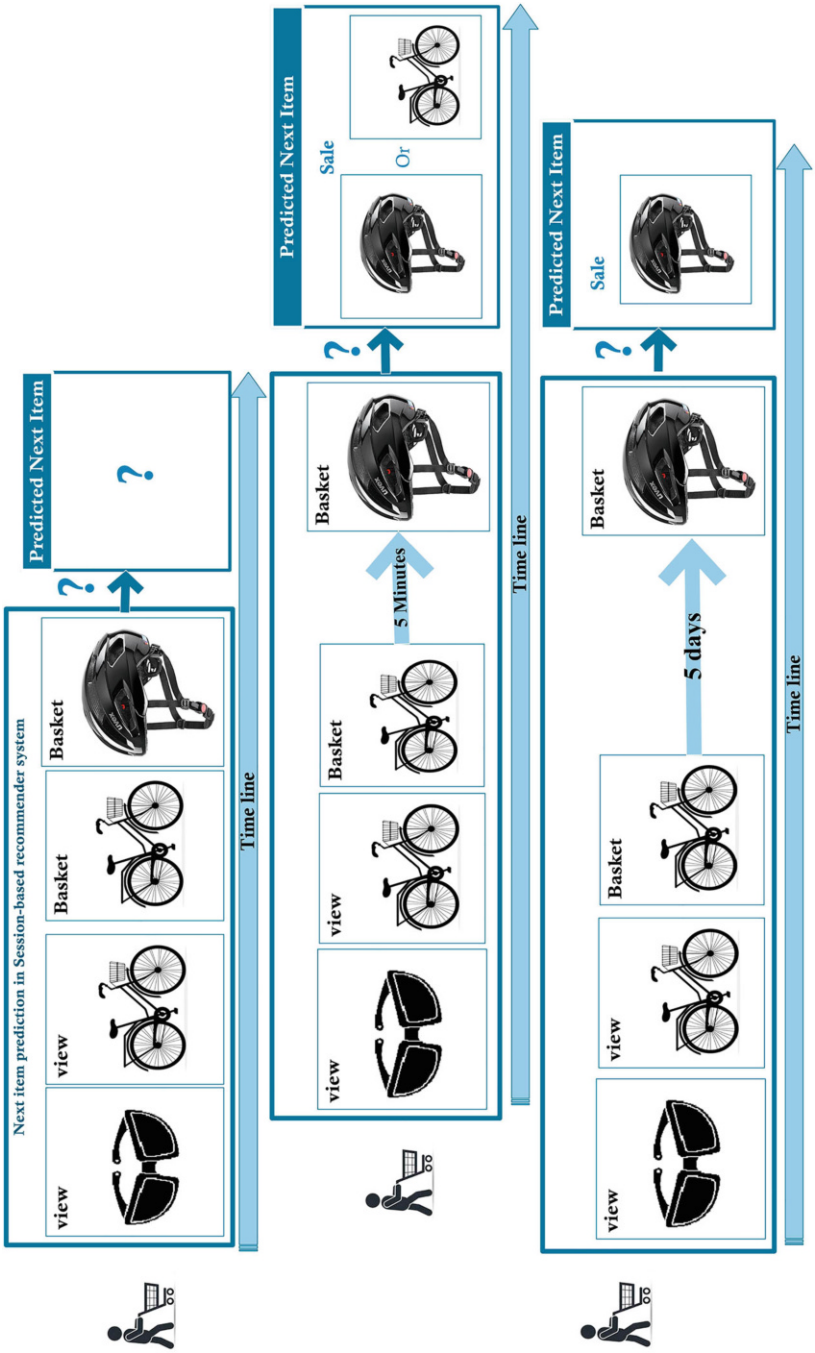

Fig. 3.4 The effect of different patterns of time gaps in the next item prediction

session or the sequence of using other sessions) [18]. Today, GRU4Rec is often used as a baseline method in empirical evaluations.

RNNs can also be utilized to model content with item features along with the click sequence interactions. By considering the features of the extracted item, such as thumbnail images of videos or a text description of a product, a parallel-RNN model was developed, which provides a better recommendation quality than the simple RNN [10]. Data augmentation techniques can also be used to improve the performance of RNNs for session-based recommendations. In these techniques, a session is divided into several sub-sessions for training, although the side effect is increasing the training time [25].

The main feature of RNN is the presence of hidden states in the units, which are not only related to the current input of the network but also related to the previous inputs. The equations related to standard RNN are as follows:

$$h_t = \sigma_h(W_i x_t + U_h h_{t-1} + b_h) \tag{3.15}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{3.16}$$

In Eqs. (3.15) and (3.16), x_t is the input vector, h_t is the hidden layer vector, y_t is the output vector, W and U are the weight matrices, and b is the bias vector.

Standard RNNs cannot learn long-term data dependencies due to the vanishing and exploding gradient issues. So, modified models of RNNs, such as GRU and LSTM, have been proposed, which have provided an effective solution to these problems by adding a gate function to the RNN network. Over time, researchers have presented various improved models of LSTM and GRU. Here, we will briefly review the relation and equations of the basic types of GRU and LSTM models.

A special type of gated RNN, namely, long short-term memory (LSTM) [58], is used in advanced SBRS to model user and item dynamics, which are dependent on the temporal data [59]. The gate mechanism is used to balance the flow of information from the current and previous time steps, so it can more effectively memorize historical information over time for an appropriate recommendation.

The LSTM standard has three gates: f_t is the forget gate that specifies how much of the previous data is to be forgotten, i_t is the input gate that evaluates the data to be stored in memory, and o_t is an output gate that decides how to compute the output based on the available data and information.

$$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i) \tag{3.17}$$

$$f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f) \tag{3.18}$$

$$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o) \tag{3.19}$$

In Eqs. (3.17), (3.18), and (3.19), parameters W, R, and b are matrices and vectors whose elements can be trained. LSTM units are defined based on Eqs. (3.20)–(3.23):

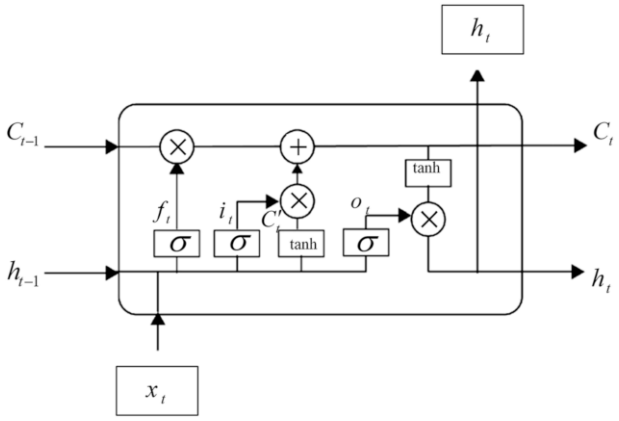

Fig. 3.5 The internal structure of the LSTM cell

$$\acute{\mathrm{C}}_t = \tan h(W_c x_t + R_c h_{t-1} + b_c) \tag{3.20}$$

$$C_t = \left(f_t \odot \mathrm{C_{t-1}} + \mathrm{i_t} \odot \acute{\mathrm{C}}_\mathrm{t}\right) \tag{3.21}$$

$$h_t = (o_t \odot \tan \mathrm{h}(\mathrm{C_t})) \tag{3.22}$$

$$y_t = \sigma\left(W_y h_t + b_y\right) \tag{3.23}$$

The state of the candidate cell $\acute{C}_t$ is calculated based on the input data x_t and the previous hidden state h_{t-1}. The cell memory or the current cell state C_t is obtained using the forget gate f_t, the previous cell state C_{t-1}, the input gate i_t, and the candidate cell state $\acute{C}_t$. The sign $\odot$ is an element-wise product. The output y_t is calculated based on the weights (W_y و b_y) corresponding to the hidden state h_t. Figure 3.5 shows the internal structure of the LSTM cell.

In particular, to enable the effective extraction of high-order temporal dynamics, gated recurrent unit (GRU) networks can be used [59]. Such networks require a more accurate model than an RNN unit, which deals with the vanishing/exploding gradient problem. GRUs are widely used in the field of RNN network applications due to the model's simplicity, reduced complexity, and computational costs compared with LSTMs. In this type of network, forget and input gates are combined to create the update gate. However, cell states and hidden states are also combined.

Briefly speaking, GRU has two gates: update gate u_t and reset gate r_t. The u_t sets the update rate of the hidden state, and the r_t decides how much of the past information is to be forgotten. Equations (3.24)–(3.28) show the basic formulations of GRU:

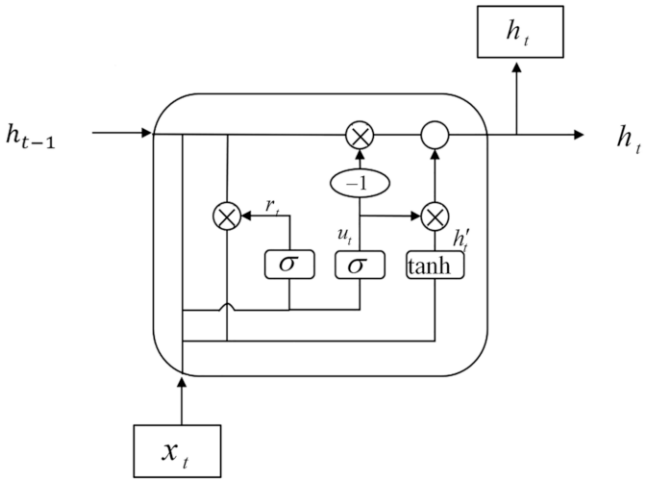

Fig. 3.6 The internal structure of the GRU cell

$$u_t = \sigma(W_u x_t + R_u h_{t-1} + b_u) \tag{3.24}$$

$$r_t = \sigma(W_r x_t + R_r h_{t-1} + b_r) \tag{3.25}$$

$$h'_t = \tan h(W_h x_t + (r_t \odot h_{t-1}) R_h + b_h) \tag{3.26}$$

$$h_t = ((1 - u_t) \odot h_{t-1} + u_t \odot h'_t) \tag{3.27}$$

$$y_t = \sigma(W_y h_t + b_y) \tag{3.28}$$

Figure 3.6 shows the internal structure of a GRU cell.

After reading the above, the question may come to mind, which model of RNN networks is better to choose for data modeling for session-based recommender systems. In several studies, such as [59], this question has been addressed and mentioned that so far, no scientific study has clearly stated the superiority of a model in a general and comprehensive manner. Although GRU creates faster models due to the lower number of parameters, LSTM can perform better if you have access to high computing power and enough input data [58, 60].

In the following two subsections, different approaches based on GRU and LSTM for session-based recommender systems will be discussed and analyzed to make the readers familiar with the pros/cons of each model.

3.3.2 GRU Approaches

The first use of deep neural networks in session-based recommender systems was in 2016, when a model based on RNN was employed [9]. This method, GRU4Rec,

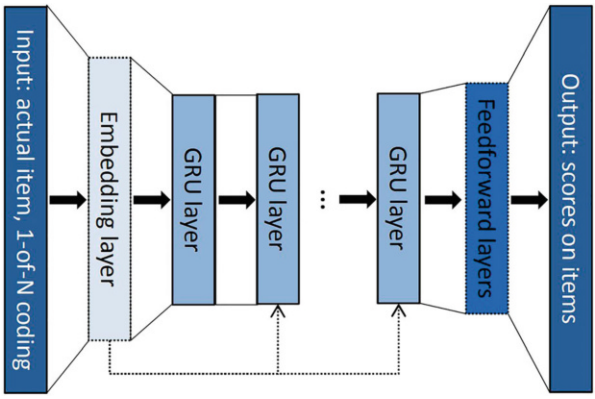

Fig. 3.7 The general architecture of GRU4Rec [9]

used a GRU network to effectively model longer sessions to solve the gradient vanishing problem.

The input data of GRU4Rec are vectors that encode items using the one-hot encoding method, and each vector's length is equal to the number of items. The output is the score predicted by GRU4Rec for a fixed number of items. In other words, the output of this method is the chance of an item being the next one in the current session. The general architecture of GRU4Rec is shown in Fig. 3.7.

Hidasi et al. developed an efficient approach called session-parallel mini-batches, which first creates an order for the sessions [9]. Then, the first events of the first x sessions create the input of the first mini-batch of length x, whose considered output is the second event of the sessions. The second mini-batch consists of the second event from the first x sessions, and their output is the third event of the sessions. If the events of any session end, then the first event of the next available session in the list of sessions will be replaced. It should be mentioned that the sessions are assumed to be independent, and when the sessions are replaced, the hidden state is reset. The view of this process is indicated in Fig. 3.8.

Calculating the score for each item in each stage is very difficult because of the large number of items. For this reason, the negative sampling method is used to calculate the score of some negative samples in addition to the resulting outputs, and the weights are updated.

In the GRU4Rec, the appropriate choice of the loss function is a critical decision that greatly impacts the quality of the recommendations. The loss functions presented in [9] are:

- Bayesian Personalized Ranking (BPR): BPR is a matrix factorization method that uses pairwise ranking loss. This method compares the score of a desirable (positive) item with the score of negative samples. In [9], the scores of positive items are compared with negative items, and their average is used as the loss. Loss at a given point of a session is calculated based on Eq. (3.29):

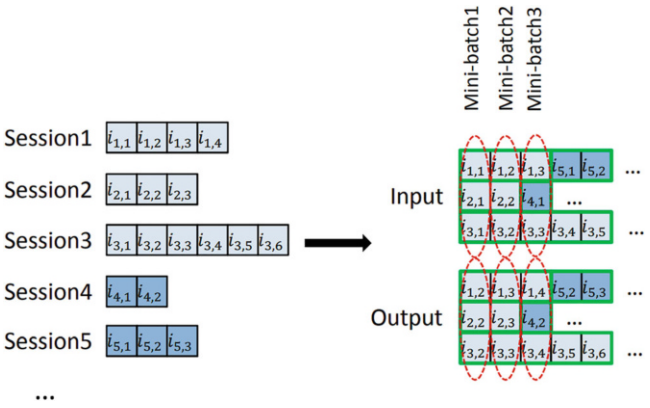

Fig. 3.8 The visual demonstration of the session-parallel mini-batches [9]

$$L_s = -\frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \log\left(\sigma\left(\hat{r}_{s,i} - \hat{r}_{s,j}\right)\right) \tag{3.29}$$

In Eq. (3.29), N_s is the number of samples, $\hat{r}_{s,k}$ is the score of item k at a given point of the session, i is the index of the desired item, and j is the index of the negative samples.

- TOP1: This function is the regularized approximation of the relative rank of the relevant item $\hat{r}_{s,i}$. Loss is calculated based on Eq. (3.30):

$$L_s(\hat{r}_{s,i}, S_N) = \frac{1}{|S_N|} \cdot \sum_{j \in S_N} \sigma\left(\hat{r}_{s,j} - \hat{r}_{s,i}\right) + \sigma\left(\hat{r}_{s,j}^2\right) \tag{3.30}$$

Using the fundamentals created in the GRU4Rec, a similar session-based recommender system was proposed by Hidasi et al. [10]. In addition to the sessions' data related to user interactions, the features of the clicked items have also been considered. Usually, in cases where there is no access to users' historical data, features such as descriptions and images of items are very effective in users' purchasing. For these reasons, Hidasi et al. use features extracted with high quality from the images and descriptions of the items, along with modeling the sessions based on deep learning techniques. Indeed, like GRU4Rec, it has modeled sessions (a sequence of user clicks) based on RNNs, but the type of network used in this method is parallel recurrent neural networks (parallel-RNN), which concurrently models the textual and visual features of the clicked items. Figure 3.9 shows the parallel-RNN architecture.

The reason for using parallel-RNNs is the inherently different nature of the input data. The features of the images are much denser than the one-hot presentation of the item ID or the BOW presentation of the text of the items. Parallel-RNNs allow each

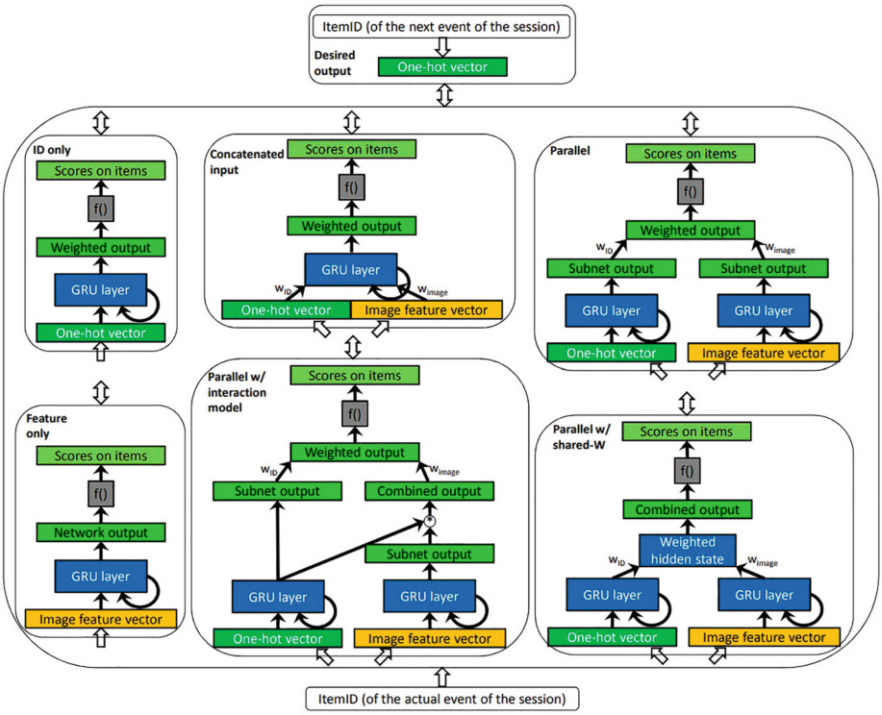

Fig. 3.9 Architecture of the parallel-RNN approach using separate GRU layers versus single GRU [10]

network to have its own configuration while maintaining communication between networks through shared parameters. In this chapter, three different configurations for parallel-RNN are presented; in the first type, each GRU is trained with one of the data representations, and the outputs are computed by concatenating the hidden layers of the subnets. In the second type, there is a shared hidden layer for the output weight matrix, where the scores for each subnet are calculated by the weighted sum of hidden states multiplied by a single weight matrix. This type has fewer parameters. In the third type, before computing the scores of each subnet, the hidden states of the subnets of the item features are multiplied by the hidden states of the subnet of the item ID. This method has been more effective than similar models that utilize fewer features along with sessions.

Hidasi et al. developed a suitable loss function that improved the results by adding different data to GRU4Rec and using more samples for the mini-batch [18]. Observations show that limiting the selection of negative samples from a mini-batch provides low flexibility. Therefore, a more efficient solution has been developed to select negative samples by sampling more items outside the mini-batch and sharing them with all mini-batches. A hyperparameter controls whether external samples are selected using uniform or popularity-based sampling. Also, a family of

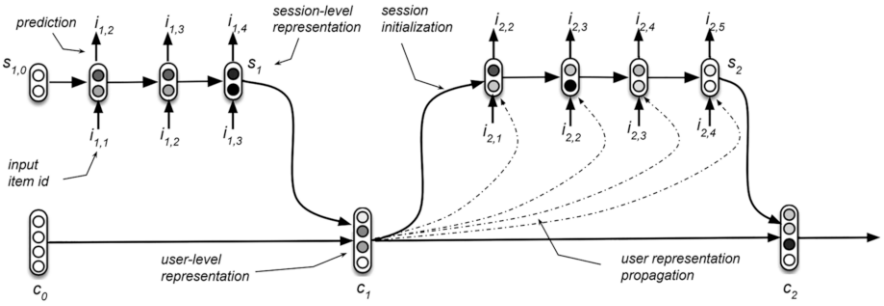

Fig. 3.10 Graphical representation of hierarchical RNN [14]

ranking-max loss functions for RNNs is used, which replaces the averaging pairwise ranking loss functions that are applied to all sampled items and the target item. This loss function calculated the loss by comparing the target item with the most relevant sample score. Moreover, this approach solves gradient vanishing by increasing the number of samples.

Another research for improving GRU4Rec is personalizing RNNs performed by Quadrana et al. through the information between user sessions, which uses a hierarchical neural network [14]. This method is based on a hierarchy of two GRUs, which are the session-level and the user-level. Session-level GRU models user interactions in sessions, and user-level GRU models and considers the evolution of user preferences across sessions over time. An illustration of this method is shown in Fig. 3.10.

In hierarchical neural networks, the hidden state of the lower-level RNN at the end of a user's session is delivered as input to a higher-level RNN. This function predicts an effective and acceptable initialization for the hidden state of the lower-level network for the next user session. The proposed method is extended by adding a GRU layer that models user activities in different sessions. It considers the changes in users' personal interests over time in providing recommendations.

Another RNN-based approach is a cross-domain SBRS which is developed by joint modeling of users' global dynamic interests and their local domain-specific behavioral sequences by exploring users' inter-session and intra-session behavioral dynamics from various domains jointly (CDHRM: cross-domain hierarchical recurrent model) [19]. Although the combination of the user behavioral changes in different domains is very effective in the quality of recommendations provided to users, it is challenging from two aspects, including the differences in behavior in different domains and asynchronous behavior. For this purpose, a cross-domain hierarchical recurrent model is proposed to integrate the sequential information of various domains, and the graphical representation is shown in Fig. 3.11.

First, a user-level cross-domain RNN that takes all the session inputs from various domains is used to determine the dynamics in the global interests of users through inter-session dynamics modeling. Then, two session-level cross-domain RNNs are developed to separately detect the intra-session dynamics in different

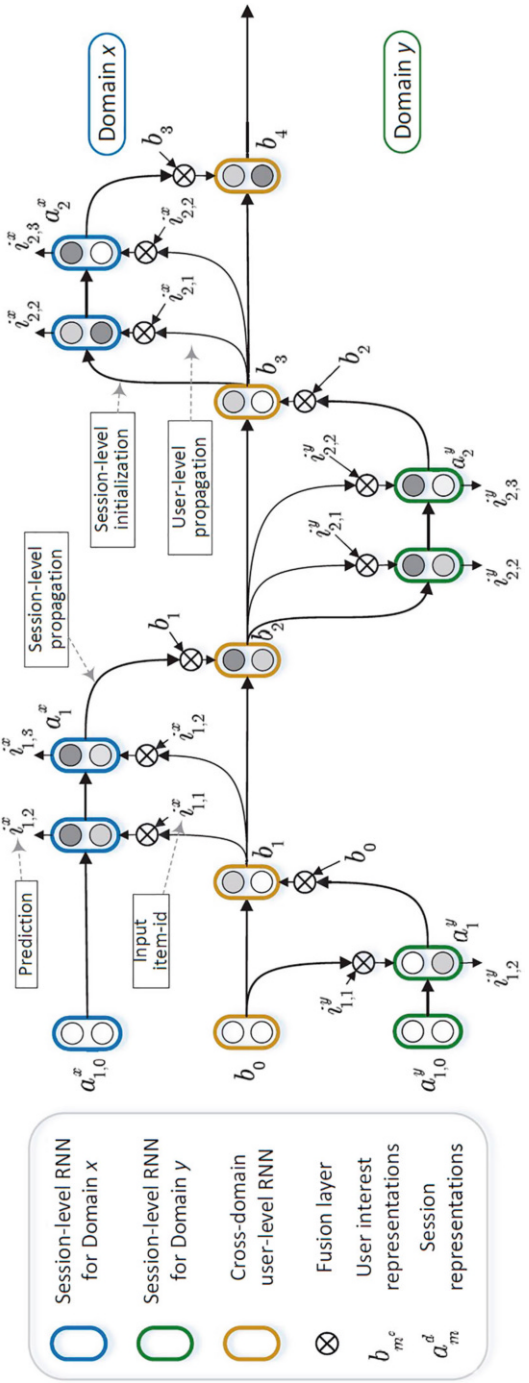

Fig. 3.11 Graphical representation of CDHRM [19]

domains. To capture asynchronous information of events in different domains, the user-level RNN allows information to be shared with the session-level RNN in the order of time. Finally, the information on the user level and the cross-domain session level are combined, and final recommendations are generated for the user.

A session-based recommender system using RNNs was developed by Hu et al., which employed user preference evolution networks that are designed in two stages (PEN4Rec: Preference Evolution Networks for session-based Recommendation) [26]. Modeling the evolution process of user interests is done based on the previous context. First, user-item behaviors are encoded in the session graph to detect complex item transfers under multi-hop neighbor connections and then extract the user's preferences through two-step retrieval. The architecture of PEN4Rec is shown in Fig. 3.12.

As shown in the above figure, in the first stage of PEN4Rec, relevant behaviors from previous behavioral contexts of recent items are integrated according to local preferences through an attention-based mechanism. The last K items are considered as of local preference because the next item may be related to the part of these K items. Relevant behaviors are then integrated through a soft attention mechanism to detect global preferences.

In the second stage of PEN4Rec, the evolution of user preferences over time is modeled dynamically, as well as the reasons for preference evolving. In the second stage, there are two key layers: the reader layer and the preference fusion layer. In the second stage, there are two main layers: the session reader layer and the preference fusion layer. In the session reader layer, using an adaptive Bi-GRU neural network, the contextual information of each item is recorded in two directions. Bidirectional GRU allows broadcast messages from neighboring contexts to obtain spatial information in the current session. The preference fusion layer uses the modified GRU neural network through the embedded information of the attention mechanism, and the internal state of the GRU is updated with the weights of the attention mechanism to increase the focus on relevant behaviors related to the preference evolution and reduce the influence of unrelevant behaviors. The advantages of this method are modeling a representation of recent preferences with relatively previous information and better predicting the next item by considering the preference's evolving trajectory.

One of the most widely used filtering approaches in recommender systems is collaborative filtering, which requires historical data to provide recommendations, and in the case of insufficient data, it will face the cold start problem [61]. A solution to overcome this problem is the session-based approach that uses the recent behavioral information of users in the current session to provide recommendations. To this end, Vikram et al. developed a deep learning-based framework called SessionRNNRec, which uses RNN to model user sessions [21]. In SessionRNNRec, sessions of a Web-based application are received as input, and events are organized through the session-parallel mini-batch method. The sessions are then modeled using the improved GRU. In this type of GRU, the activation function is selected based on a linear interpolation between the candidate and the previous activation functions

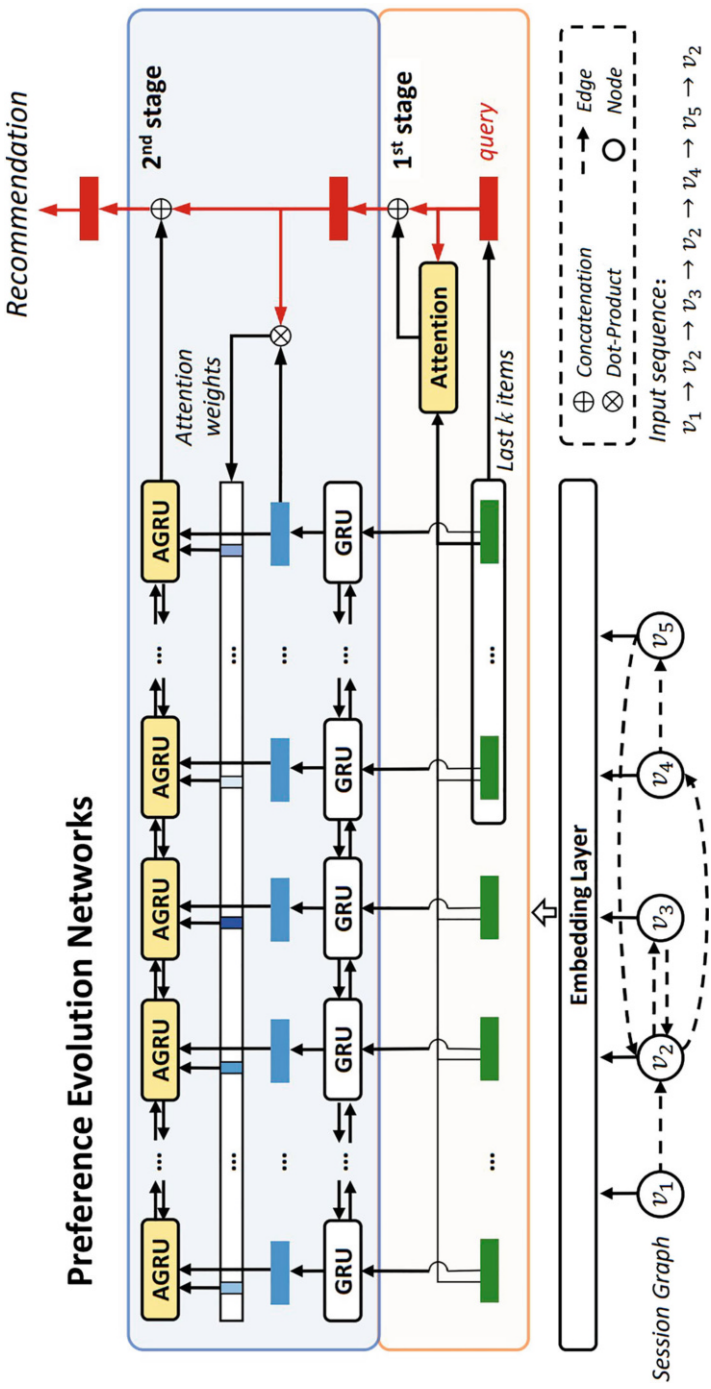

Fig. 3.12 Architecture of PEN4Rec [26]

and is trained using mini-batches. The recommendations generated in SessionRNNRec are actually the next item in each session.

The sparsity of sequential data and the lack of data needed to provide recommendations to users have led to the proposal of a new context-based recommender system that places contextual information along with sparse sequential data [23]. The input of the network is the sessions, and the output is the next item in the session. This method consists of four steps; in the first step, the contextual information of the session, which includes the country and type of user's device, is encoded by the one-hot method. Then, the encoded vectors are mapped from the vector space with high dimensions to dense numerical vectors with low dimensions, facilitating the extraction and summarization of features by a neural network. Then, the data are combined with one of the Add, MLP, and Stack. In the Add method, the vectors are superimposed directly, their dimensions do not increase, but the dimensions of the vectors must be equal. In the Stack method, vectors are placed on top of each other, and the dimensions of the input or output of the neural network increase and require more computational resources for processing. The last solution is to use MLP networks, which create a combined vector based on the calculation of a weight matrix. The disadvantages of this solution are the need for many computational resources and the difficulty of the learning process. Then, in the next step, the vectors are fused into the GRU network and are processed and modeled by this network. It is recommended that the initialization of GRU hidden states should be done at the beginning of each session so that they require fewer computational resources.

3.3.3 LSTM Approaches

In session-based recommender systems using deep learning, there has been much attention and acceptance toward LSTM networks because these types of deep neural networks effectively and optimally provide recommendations to users. These recommendations often deliver more precise results than the previous sequential methods.

Lenz et al. presented an LSTM-based approach that generates a representation of items in the vector space [31]. Vector space representations of items are used as input to recommender systems. Learning vector space representations of items allows the relations between items to be correctly recognized. Unlike other vector space presentations that use an additional network for learning, this method does not need to pre-train the features of the items and can be trained end to end. By using this embedding method, the items are presented in a continuous vector space with high dimensions, which can represent multiple relations for a product. In contrast to this, LSTM-based embedding method is the one-hot method, which creates vectors for presenting items that are very sparse. Also, the proposed method greatly reduces computation complexity compared with the one-hot embedding method.

Dobrovolny et al. investigated the use of the LSTM network as a deep learning technique for session-based recommender systems [38]. This study proposes a

solution for using word-level LSTM as a real-time recommendation service. To make the input data suitable for modeling by LSTM, the data should be transformed in a sequence of steps. For example, if there is a list of news that the user has liked, it can be converted from a list to a sequence and, for each user, considered a sentence containing the ID of the news liked by him/her. In this method, each user has a history of likes in a certain period, based on which the next news liked by the user can be predicted. The session-based proposed system in [38] consists of three main components of embedding, LSTM network, and dense layer. In the word embedding stage, feature learning techniques are used to convert words into numerical vectors. In the next step, two layers of LSTM are used to process and model the data. The dense layer is designed to the number of output classes that predict what the next class is to recommend to the user. To improve this method that was proposed at the word level, a method presented in [36] performs embedding at the character level, which can process a larger dataset.

Utilizing LSTM as the main model of the recommender system requires sufficient data to be available, and if possible, there should be access to relations between classes. Using LSTM for small datasets leads to overfitting or generating a weak and inefficient model. Therefore, the use of additional information than session data can improve accuracy in session-based recommender systems. In [35], to detect the dynamic interests of software developers (users), their dependency constraints and social influence have been utilized and provided a Session-based Social and Dependency-aware software Recommendation (SSDRec) system. This system works based on the social influence of the user and the dependency between the software packages (items), which consists of two attention-based graph networks. These networks are, respectively, used to detect the dependencies between the items and the social influence of the users, and a recurrent LSTM neural network for modeling the short-term dynamic interests of developers in each session is formed. The architecture of SSDRec is presented in Fig. 3.13.

As shown in Fig. 3.13, SSDRec consists of four main components:

1. Dependency constraint: By using an attention-based graph network, dependency relationships between software packages are detected, and a presentation vector is embedded for each software package.
2. Dynamic interest modeling: An LSTM recurrent neural network is used to model the order of software packages in a session and obtain the embedded vector presentation of the dynamics of each software developer's interests in each session. The inputs of the LSTM network are the sessions related to the software developer and the presentation vectors of the software packages, which sessions include a time-ordered set of software packages viewed by each software developer. The LSTM network recursively learns a latent representation from the order of the current input and previous input software packages.
3. Social influence detection: In this component, a graph attention network is used to obtain the social influence of each developer based on other neighbor developers who are friends. Based on this graph, embedded vectors are created to represent each developer.

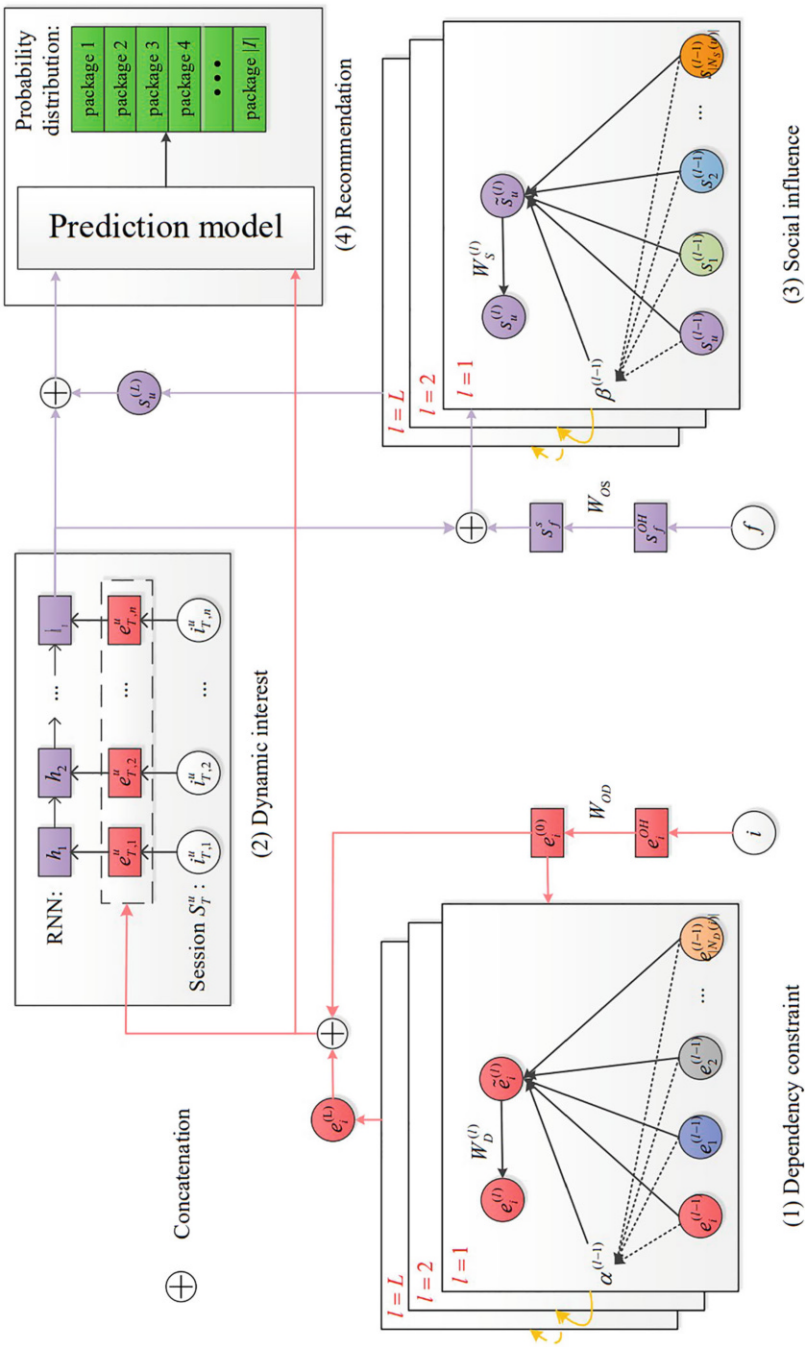

Fig. 3.13 Architecture of SSDRec [35]

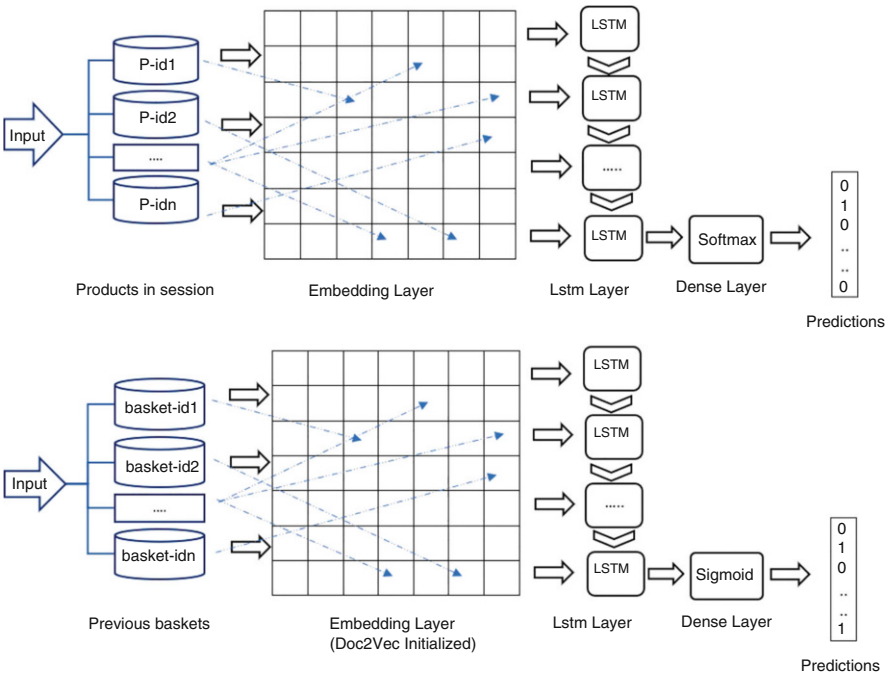

Fig. 3.14 Next-item (top) and last-basket (bottom) SBRS architecture [34]

4. Recommendation: The probability of selecting a software package by a developer is estimated using a softmax function.

Salampasis et al. proposed two recommender systems, one of which is completely content-based and utilizes the Doc2Vec model to generate the vector of item representation based on the text description of each item [34]. The second recommender system utilizes the Item2Vec method to generate a vector of items, which operates based on item-based collaborative filtering. This method of vector representation and embedding of items are used to infer item-to-item relations and usually used in session-based recommender systems. Finally, in this article, a combined embedding method based on Doc2Vec and Item2Vec is proposed, which takes into account the pattern orders between items in addition to the content of the items. The LSTM recurrent neural network has been considered as the core component in the proposed approach, in which the inputs of the LSTM are generated in each system by one of the various embedding methods listed above. Figure 3.14 shows the architecture of next-item and last-basket SBRS.

In [34], in addition to choosing the type of embedding method and vector representation of items, two tasks of next-item recommendation and next-basket recommendation were investigated. LSTM presents the more efficient performance in the next-item recommender system that uses the content-based method, but it is

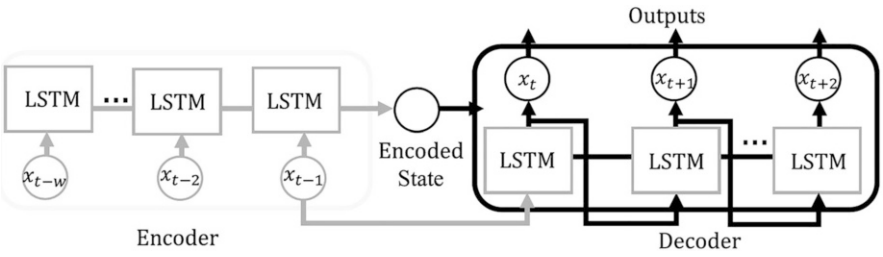

Fig. 3.15 Architecture of DeepCBPP [37]

less efficient in the basket. Evaluation results show that the combination of Doc2Vec and Item2Vec together with the LSTM network does not provide better results.

In many recently published research, the time intervals have been considered as an explicit component and used in the learning process of sessions. However, the effects of multiple previous levels are not considered; instead, a sequence with a limited length is considered. For this purpose, Fuentes et al. modeled the sequential prediction problem as a multi-class classification based on LSTM [37]. The Deep Customer Buying Preference Prediction (DeepCBPP) method automatically learns behavioral patterns from the history of purchase transactions and predicts the next purchase item or the category to which the next item belongs. The architecture of DeepCBPP consists of four parts: transactional data, the customer sequences file, the LSTM training model input, and the output of the training model. These components form the inputs and outputs of the three stages of the DeepCBPP, which are customer buying sequence transformation, multi-level preference generation, and preference buying learning.

An LSTM layer is capable of learning temporal dependencies, but a chain of LSTMs is more suitable for processing time-based sequential data. For this purpose, a combination of encoder-decoder and stacked LSTMs is used in DeepCBPP, as shown in Fig. 3.15. The LSTM encoder processes a customer preference input and generates an encoded state. The LSTM decoder uses the encoded state to produce an output. Evaluations of DeepCBPP show improved accuracy and stability of recommender system performance with multilayer LSTMs. In fact, a new sequence customer presentation method is presented as the basis of the data transformation process, which allows processing with multi-level interactions in scenarios where the length of sequences may be limited and the interactions have more dependencies on previous sessions.

3.4 Session-Based Recommender System Using CNN

Before looking at the approaches of convolutional neural network models in session-based recommender systems, an overview of CNN and the reasons that made it an effective choice for SBRS are provided.

3.4.1 Why CNN?

Convolutional neural network (CNN) is a type of neural network architecture that has achieved advanced results in machine vision, speech recognition, and NLP. By applying convolution operations (known as kernels or filters) at different levels of granularity, the CNN model can extract useful spatial and temporal features from the data learning tasks and reduce the need for manual feature engineering. This particular feature is much needed in SBRS because the goal is to extract useful patterns from the flow of clicks and predict the next events. A pattern can be a sequence of clicks, a sequence of categories related to products or their names, or a specific combination of all mentioned items.

The CNNs are useful for processing user sessions in two ways: (1) The sequence of items in one session or between different sessions of users can be easily implemented and modeled on CNNs. (2) They have a high capacity to learn local features of a segmented area or specific relations between different areas, based on which they can recognize dependencies that other models usually ignore.

Generally, in session-based recommender systems, to learn and model data related to users and items, the input data must be embedded suitably to use convolution and pooling layers to detect temporal and spatial patterns between them. The user's favorite items are predicted based on the features obtained from the input data and the dependencies between them. For this purpose, suppose each interaction is represented by a d-dimensional vector and the embedding matrix of each session (which includes n interactions) is considered as $E \in R^{d \times |c|}$. Then, in a horizontal convolutional layer, by convolving the x-th filter F^x from the top to the end of the E matrix, the value of a_m^x is obtained as the following equation (in the m-th convolution based on the x-th filter):

$$a_m^x = \varphi_\alpha(E_{m:m+h-1} \odot \mathrm{F}^{\mathrm{x}}) \tag{3.31}$$

In Eq. (3.31), φ_α specifies the activation function for the convolutional layer.

The final output $e_c \in R^z$ from the z filters is obtained based on applying the max pooling on the convolution results $\alpha^x = \left[\alpha_1^x, \alpha_2^x, \ldots, \alpha_{|c|-h+1}^x\right]$, with the aim of obtaining the most significant features of a session using Eq. (3.32):

$$e_c = \max\left\{\max\left(\alpha^1\right), \max\left(\alpha^2\right), \ldots, \max\left(\alpha^z\right)\right\} \tag{3.32}$$

Indeed, e_c is a presentation of the interaction of a session. The general architecture of CNNs for session-based recommendation is shown in Fig. 3.16. In this figure, x_1 to x_{11} represent item embedding vectors of a session as one-dimensional vectors, and convolution layers are applied to these data to scan and check the session data.

In contrast to RNNs, the training of CNNs does not depend on previous timestamp calculations and therefore allows parallelization on each element in a sequence.

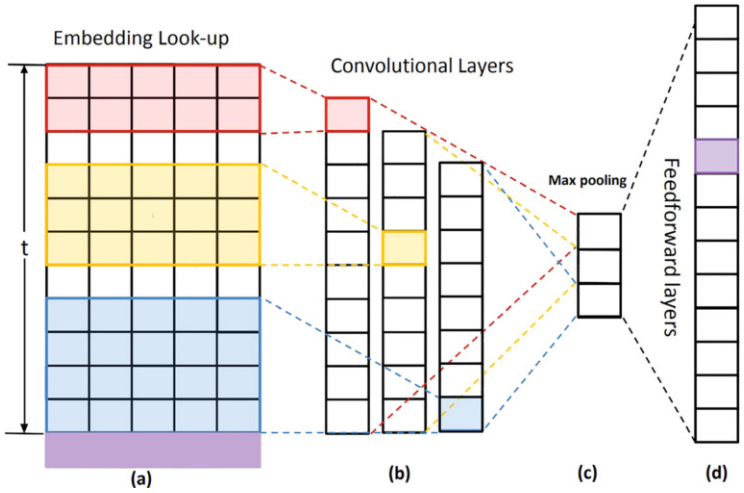

Fig. 3.16 Different layers of CNN in a session-based recommendation [41]

3.4.2 CNN Approaches

Tuan et al. were among the pioneers of using CNNs in the field of session-based recommender systems and presented a method based on character-level embedding and 3D-CNN [40]. The main difference between two-dimensional CNNs (2D-CNN) and three-dimensional CNNs (3D-CNN) is that in the 3D-CNN, convolution and pooling operations are performed in all three dimensions and on all three axes of the cubic data structure, but in 2D-CNNs, the operation is performed only on two dimensions. Models that use character-level embedding to represent concepts and convert them into numerical representations can easily model different types of data and perform the feature engineering stage.

Each input feature based on an alphanumeric format (combination of alphabet letters from a to z, numeric digits from 0 to 9, and some special characters such as @, $, -, etc.) becomes a vector without the need for classified embedding. Therefore, each item is considered as a two-dimensional structure: one dimension is the features, and the other one is the characters. They are represented in the form of a three-dimensional structure based on the order of the interactions of each session, which is actually the chronological order of the user's visit to the items. Accordingly, to obtain the spatial and temporal features of the input data, several 3D convolution layers are used in a stack, which is applied between a feature map and a 3D kernel of the 3D convolution operator at each step. For each convolution layer, there are several kernels, and the results of kernels are feature maps to use in subsequent layers.

Moreover, the residual connections are used in [40]. There are two types of residual connection: the identity connection is applied to inputs and outputs with the same dimension, and the projection connection is used to reduce distance

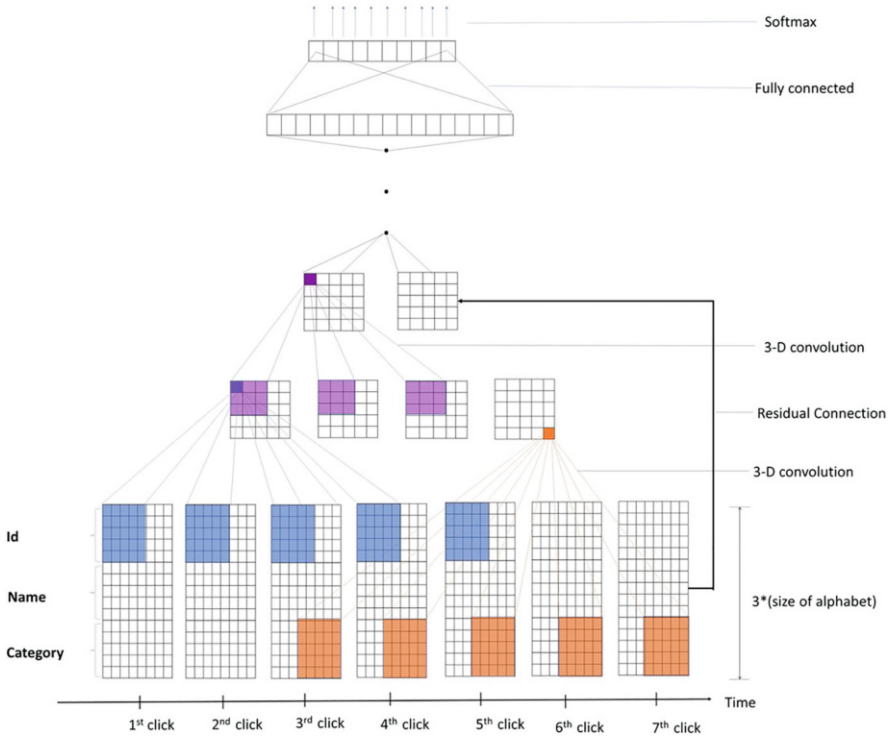

Fig. 3.17 Graphical representation of 3D-CNN SBRS [40]

samples through the implementation of 1*1 convolution with a step size of 2. The output represents a vector with the length of the number of items, which shows the probability of their selection by the user. The general architecture 3D-CNN SBRS is shown in Fig. 3.17.

The sequential convolution technique is presented to recommend the next item by Yuan et al. to overcome the challenges of increasing the length of sessions and the number of interactions in session-based recommender systems [41]. In this chapter, a simple and basic method is presented that allows complex conditional distributions to be modeled despite very long-range sequences. This model first explicitly encodes the dependency between items in such a way that the distribution of the output sequence is estimated. Then, instead of using large inefficient filters, it stacks dilated one-dimensional convolutional layers on top of each other to increase the receptive fields when modeling long-term dependencies.

Dilated CNN has been utilized for prediction in the fields of image generation, translation, audio, etc.; however, it had been unused in the field of recommender systems until the publication of this chapter. To optimize this deep architecture, residual networks are used to cover convolutional layers with residual blocks. To create the input matrix of this method, the convolutional neural network stores user-

item interactions in a matrix and considers the matrix as an image in latent space. Another point of this method is to propose a masking-based dropout trick for one-dimensional dilated convolution to overcome the problem of information leakage, which prevents the network from seeing future items.

Tang et al. used CNNs to learn sequential features and employed the latent factor model to learn user-specific features [42]. The purpose of the multilayer Caser (Convolutional Sequence Embedding Recommendation Model) network is to detect the user's general interests and sequential patterns on both union and point levels and to see the user's skip behaviors in unobserved spaces.

As shown in Fig. 3.18, Caser consists of three components: an embedding lookup table, convolutional layers, and fully connected layers. To train the layers of convolutional neural networks for each user u, L, successive item, and T, the next item, are extracted from the sequence S^u as the input, which is shown on the left side of Fig. 3.18. This is achieved by sliding a window of length $L+T$ over the user's sequence, and each window is a training sample for user u, represented by the triple (u, previous L items, next T items).

Yuan et al. improved the dilated CNN proposed in [41] by utilizing a gap-filling encoder-decoder framework using masked convolution operators that provides the ability to simultaneously consider data from past and future contexts without data leakage [35]. In this method, the encoder takes the partially complete session sequence as an input, and the decoder predicts the masked items based on the encoded representation. In the proposed method, the encoder should be aware of the user's general interests represented by the unmasked actions simultaneously, and the decoder predicts the next item based on the user's previous contexts and encoded general user interests.

The convolutional neural networks with sparse kernels used in [44] have two main advantages: (1) providing an autoregressive mechanism to create sequences and (2) creating two-side contexts for encoding. Moreover, the projector neural network proposed in this chapter increases the representational bandwidth between the encoder and decoder. The encoder is implemented with a set of stacked one-dimensional dilated CNNs that both dilated layers are covered with a residual block to avoid the gradient vanishing. The decoder also consists of embedding layers, the projector, and casual CNNs.

A special type of convolutional network (TCN: temporal convolutional network) has been used in some recent models that can detect the dependency between non-adjacent items in a session and balance the gaps.

In this regard, Ye et al. proposed a session-based recommender system that uses cross-session information in addition to information within sessions to provide recommended items (CA-TCN: cross-session aware temporal convolutional network) [46]. There are two types of cross-session information, items in another session that have common features with the current item and the context of the session, which specifies the interests of users similar to the current user of the session. At the item level, a directed item graph is created between global to consider the effect of mutual session on each item. At the session level, a graph is created for

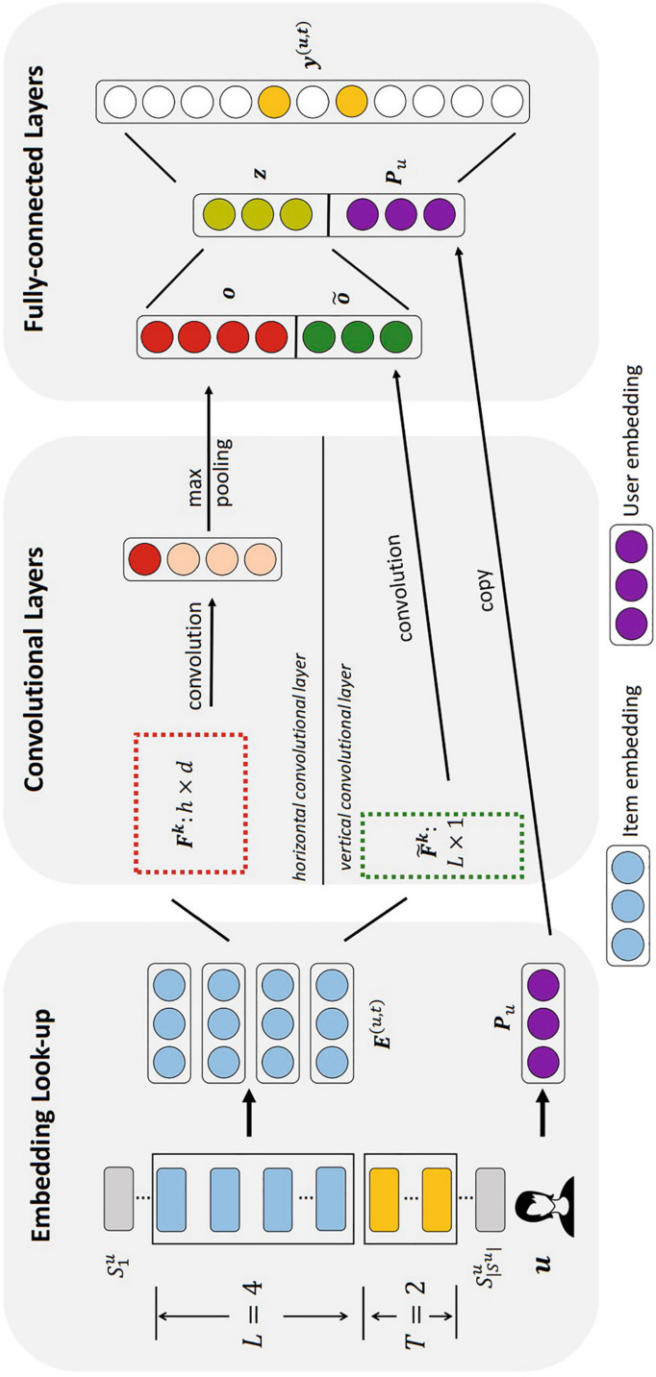

Fig. 3.18 Architecture of Caser [42]

cross-sessions, whose each edge represents the degree of similarity between two sessions.

A TCN is a sequence of convolutional models in which sequential information is unlost during convolution operations because the same items have various representations at different times. TCN can use the direct effect between items and solve the problem of long-term dependencies and lack of sequential data. In fact, by using TCN, convolution operations can be performed on long-term items, and sequence information can be involved in the process of this operation. In this chapter, in fact, a model based on hierarchical attention at the item level and at the session level has been proposed, in which the influence of items and sessions are considered simultaneously.

Although the proposed methods in session-based recommender systems using deep learning methods have obtained efficient results, there are still two basic challenges: (1) The value of each dimension in the results of the embedding layer is distributed with a non-zero mean, and very large numerical gaps increase the variance of the gradient, impede the optimization of the parameters, and ultimately lead to incorrect recommendations. (2) Previous models cannot effectively and correctly learn information about long-term dependencies and cannot recognize dependencies between non-adjacent items in a session.

In [45], to solve these problems, another type of temporal convolutional neural network is used to balance numerical gaps. In this method, the results of the embedding layer are first normalized, and then the results obtained in the unit hypersphere are limited to reduce their effect on the gradient calculation. Finally, the use of the TCN completes the multilayer self-attention network to learn session order.

3.5 Discussion

The proposed approaches discussed in this chapter provide different models for session-based recommender systems utilizing one of the deep discriminative techniques, such as GRU, LSTM, CNN, and variations of CNN. Due to the sequential nature of session data, the majority of works use GRU and LSTM recurrent neural networks. By proposing the GRU4Rec approach, a new roadmap was developed for the use of RNNs in session-based recommender systems. Furthermore, other methods such as [9, 14, 18] have been developed to improve GRU4Rec recently. The GRU4Rec employs the session-parallel mini-batch technique to accelerate the learning process, which is the key advantage of this method and similar ones.

However, a limitation of the GRU4Rec in new contexts is that the model can only recommend items in the training set because this model was trained to predict the scores in a limited number of items. Moreover, RNN-based approaches, such as [16], may not improve the performance and prediction accuracy compared to simpler methods due to the use of item identifiers in the learning process without considering any other side information [55]. To solve such problems, the recently proposed

methods in this field use additional data and other deep learning techniques to extract features and embed them to item and user representation so that they can consider their various features in analysis and modeling and recommend items with high accuracy.

The methods such as [9, 14, 18, 42] do not consider parts of the item's features, which may lead to the incorrect detection of user interests in situations where contextual data is ignored. Therefore, as mentioned above, additional information helps to improve the performance of session-based recommender systems. To this end, research such as [10] has utilized GRU4Rec with additional information such as images and descriptions of images along with item identifiers. This improves system performance and reduces the cold start problem. Furthermore, the evaluation results of [23] also demonstrate the impact of contextual data in improving the performance of the session-based recommender system. In addition, increasing the rank of the more relevant items, reducing the effect of noisy data, and thus increasing the model's stability are advantages of using contextual data [23].

Another way to increase the amount of data is to use user and item data in various contexts. For example, the authors in [19] also use information from several domains to provide useful recommendations. Although the consideration of user behavior changes in different domains is effective in the quality of recommendations, two major challenges should be taken into account: the behavior differences in various domains and the asynchronous behavior.

Another improvement of GRU4Rec has been performed in [14] by using an additional GRU level that utilizes a hierarchy of GRU networks, which considers the dependencies within each session and the dependencies between sessions. Although this method has achieved some successes, it does not consider the randomness of user interactions, and as a result, it may not correctly predict the user's current goal in some situations.

Some SBRS, such as [31, 34–38], employ LSTM networks to model data. For example, in [31], LSTM is used for embedding information, and items are presented in a continuous vector space with high dimensions, which can cover several relations. On the opposite side of this LSTM-based embedding method is the one-hot method, which represents sparse vectors for item embedding. However, the computation complexity is greatly reduced compared to the one-hot embedding. Several methods, such as [36] and [38], have also used character-level LSTM and word-level LSTM for data modeling. It should be mentioned that the word-level LSTM technique can only be used for small-size datasets and the character-level LSTM has been efficient for larger datasets.

Considering the advantages of using additional data for better modeling, the authors in [35] have obtained suitable results using the social data of users and the relations between them based on LSTM and graphs. The efficiency of this method is less for long sessions because short sessions include short-term dynamic interests, but long sessions represent long-term static interests.

Using LSTM and the content-based method to recommend the next item leads to efficient performance [34], but it has a weaker performance for the next-basket recommender system. Investigations and results show that the combination of

embedding methods such as Doc2Vec and Item2Vec and using them along with the LSTM network does not provide better results.

In several methods, the improved classic mode of networks is used, such as stacked LSTMs in [37]. The evaluation results show an improvement in the accuracy and stability of the recommender system's performance. This method allows working with multi-level interactions in scenarios where sequences may be short in length and interactions have more dependencies on previous sessions. But this method is significantly complicated due to the nature of the stacked LSTM.

Usually, RNNs have been considered the most effective technique for sequential data. However, the effectiveness of CNN-based methods represents that CNNs are also a suitable architecture for modeling sequential data, especially when sequence elements are associated with complex features. In some research related to this field, the standard model of CNNs has been used [42], but in some other methods, such as [40, 41, 44–46], improved types of CNNs such as 3D-CNN, dilated CNN, and temporal CNN have been used. By using 3D-CNNs, the temporal and spatial features between the data are simultaneously extracted and modeled according to the sequence of the data in the sessions. Moreover, instead of using one-hot vectors, the character-level embedding method is used, which requires a less number of parameters, but the input tensor of this method has a fixed size, so there is a limit for the length of the text data and the maximum number of session clicks.

Dilated one-dimensional convolutional layers used in [41] and [44], when stacked on top of each other, increase the receptive fields when modeling long-term dependencies. Moreover, they propose a masking-based dropout trick for one-dimensional dilated convolution to overcome the problem of information leakage, which prevents the network from seeing future items.

Several approaches use temporal CNN, such as [45, 46], which can model the direct effect between items and solve the problem of long-term dependencies and lack of sequential data. In addition, the use of cross-sessions makes it possible to consider its effect on each item.

Single-domain session-based recommender systems deal exclusively with a specific domain while ignoring the user's interest in other domains and intensifying the challenges of cold start and sparsity. Solutions to such problems can be provided using cross-domain recommendation, which typically exploits the knowledge learned from the domains and produces the target recommendation. One of these approaches can be based on transfer learning, which utilizes the knowledge obtained from one domain to improve learning tasks in another domain.

Session-based recommender systems based on multi-task learning can provide better performance compared to single-task learning. An advantage of using multi-task learning in a deep neural network is its ability to reduce the problem of data sparsity through implicit data augmentation. Another advantage is that learning many tasks at the same time can prevent overfitting by simplifying the shared hidden representation.

An attention mechanism is a technique that enables a neural network to focus on a subset of features by selecting a specific input. This mechanism can be directly applied to many deep learning architectures, such as CNN and RNN. The main goal

Table 3.5 A summary of the reviewed research

Ref.	Domain	Deep learning model	Input data	Embedding technique	Loss function
[9]	Video, e-commerce	GRU	Items of sessions	One-hot encoding	BPR, TOP1
[10]	Video, e-commerce	GRU	Text description, ID and image of session items	Bag-of-words and TF-IDF + one-hot encoding + CNN	TOP1
[14]	Job, video	GRU	Items of sessions	One-hot encoding	TOP1
[18]	Video, e-commerce	GRU	Items of sessions	One-hot encoding	BPR-max, TOP1-max
[19]	Movie, music, book	GRU	User interest, sessions, and items	One-hot encoding	A weighted loss function based on TOP1-max
[21]	Video, e-commerce	GRU	Sessions, interactions of sessions	One-hot encoding	BPR, TOP1
[23]	Movie, news	GRU	Contextual information of session, items of sessions	One-hot encoding + random distribution low-dimensional vector	BPR, cross-entropy
[26]	Music, e-commerce	GRU	Items of sessions, sessions	One-hot encoding + d-dimensional node vector of session graph (GGNN)	Cross-entropy
[31]	E-commerce	LSTM	Items of sessions	D-dimensional vector	Cross-entropy
[34]	E-commerce	LSTM	Items of sessions	Doc2Vec/Item2Vec	Cross-entropy
[35]	GitHub	LSTM	Items, users, sessions	One-hot encoding + graph attention network	Log-likelihood
[37]	E-commerce	Stacked LSTM	Sessions, interactions of sessions	One-hot encoding + encoder-decoder	Cross-entropy
[40]	E-commerce	3D-CNN	Sessions, interactions of sessions	Character-level embedding	Cross-entropy
[41]	Music, e-commerce	Dilated CNN	Items of sessions	One-hot encoding + 1D convolutional filters	Binary cross-entropy
[42]	POI, movie, e-commerce	CNN	Interactions of sessions	Embedding lookup	Binary cross-entropy
[44]	Movie	Dilated CNN	Sessions	One-hot encoding	Cross-entropy
[45]	E-commerce	Temporal CNN	Items of sessions	D-dimensional vector	Cross-entropy
[46]	E-commerce	CNN	Interactions and items of sessions	One-hot encoding + GNN	Cross-entropy

of the attention technique is to provide a solution to better remember network inputs. For example, attention techniques applied to the CNN model help the model absorb the most useful elements of input information. The attention-based RNN model also enables the model to process noisy inputs. It also helps the LSTM remember input elements when handling long-range dependencies.

Table 3.5 summarizes the existing works discussed in this chapter and addresses the application domain, deep learning model, type of input data, embedding technique, and loss function of each approach.

3.6 Conclusion

In this chapter, different approaches to deep discriminative models in session-based recommender systems have been discussed and analyzed regarding the models, datasets, evaluations, and highlights/limitations of each. Various applications have been addressed in these approaches, such as e-commerce, movies, news, books, etc. Because of the sequential nature of session data, many of the proposed methods utilized RNNs, including GRU and LSTM, which can detect dependencies and relations between data and predict the relevant next item efficiently. Indeed, the ability to model the dynamic behavior of users over time in session-based recommender system has made RNNs an appropriate solution in this scope. Both GRU and LSTM networks provide appropriate results and eliminate the vanishing/exploding gradient issue. However, GRU networks have less computational complexity due to the less number of gates and parameters, while LSTM networks could provide more accurate results.

In addition to the high performance of recurrent neural networks in session-based recommender systems, the temporal and spatial features of the session data can be efficiently extracted using the standard and improved types of CNNs, such as 3D-CNN, dilated CNN, and temporal CNN. The sequence of items in one session or between different sessions of users can be easily implemented and modeled on CNN. Moreover, CNNs have a high capacity to learn the local and spatial features of regions and capture the related dependencies that are usually ignored by other models.

This chapter concluded with several discussions on the reviewed research and provided future directions and trends in session-based recommender systems using deep discriminative models.

References

1. Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal processing magazine 29, no. 6 (2012): 82-97. https://doi.org/10.1109/MSP.2012.2205597
2. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang et al. "Imagenet large scale visual recognition challenge." International journal of computer vision 115 (2015): 211-252. https://doi.org/10.1007/s11263-015-0816-y
3. Malte Ludewig, and Dietmar Jannach. "Evaluation of session-based recommendation algorithms." User Modeling and User-Adapted Interaction 28 (2018): 331-390. https://doi.org/10.1007/s11257-018-9209-6
4. Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. "Performance comparison of neural and non-neural approaches to session-based recommendation." In Proceedings of the 13th ACM conference on recommender systems, pp. 462-466. 2019. https://doi.org/10.1145/3298689.3347041
5. Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. "Deep learning based recommender system: A survey and new perspectives." ACM computing surveys (CSUR) 52, no. 1 (2019): 1-38. https://doi.org/10.1145/3285029
6. Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. "DKN: Deep knowledge-aware network for news recommendation." In Proceedings of the 2018 world wide web conference, Lyon, France, April 23 - 27, 2018, pp. 1835-1844. https://doi.org/10.1145/3178876.3186175
7. Tran Khanh Dang, Quang Phu Nguyen, and Van Sinh Nguyen. "A study of deep learning-based approaches for session-based recommendation systems." SN Computer Science 1 (2020): 1-13. https://doi.org/10.1007/s42979-020-00222-y
8. Li Deng, and Navdeep Jaitly. "Deep discriminative and generative models for speech pattern recognition." In Handbook of pattern recognition and computer vision, pp. 27-52. 2016. https://doi.org/10.1142/9789814656535_0002
9. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In Proceedings International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016. https://doi.org/10.48550/arXiv.1511.06939
10. Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. "Parallel recurrent neural network architectures for feature-rich session-based recommendations." In Proceedings of the 10th ACM conference on recommender systems, Boston Massachusetts USA September 15 - 19, 2016, pp. 241-248. https://doi.org/10.1145/2959100.2959167
11. Elena Smirnova, and Flavian Vasile. "Contextual sequence modeling for recommendation with recurrent neural networks." In Proceedings of the 2nd workshop on deep learning for recommender systems, Como, Italy, Aug 27-31, 2017, pp. 2-9. https://doi.org/10.1145/3125486.3125488
12. Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. "Embedding-based news recommendation for millions of users." In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, Halifax, NS, Canada, August 13-17, 2017, pp. 1933-1942. https://doi.org/10.1145/3097983.3098108
13. Alexander Dallmann, Alexander Grimm, Christian Pölitz, Daniel Zoller, and Andreas Hotho. "Improving session recommendation with recurrent neural networks by exploiting dwell time." arXiv preprint arXiv:1706.10231 (2017). https://doi.org/10.48550/arXiv.1706.10231
14. Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. "Personalizing session-based recommendations with hierarchical recurrent neural networks." In proceedings of the Eleventh ACM Conference on Recommender Systems, Como, Italy, Aug 27-31, 2017, pp. 130-137. https://doi.org/10.1145/3109859.3109896
15. Massimiliano Ruocco, Ole Steinar Lillestøl Skrede, and Helge Langseth. "Inter-session modeling for session-based recommendation." In Proceedings of the 2nd Workshop on Deep Learning

for Recommender Systems, Como, Italy, Aug 27-31, 2017, pp. 24-31. https://doi.org/10.1145/3125486.3125491
16. Dietmar Jannach, and Malte Ludewig. "When recurrent neural networks meet the neighborhood for session-based recommendation." In Proceedings of the eleventh ACM conference on recommender systems, Como, Italy, Aug 27-31, 2017, pp. 306-310. https://doi.org/10.1145/3109859.3109872
17. Yu Sun, Peize Zhao, and Honggang Zhang. "Ta4rec: Recurrent neural networks with time attention factors for session-based recommendations." In 2018 international joint conference on neural networks (IJCNN), Rio de Janeiro, Brazil, July 8-13, 2018, pp. 1-7. https://doi.org/10.1109/IJCNN.2018.8489591
18. Hidasi Balázs, and Alexandros Karatzoglou. "Recurrent neural networks with top-k gains for session-based recommendations." In Proceedings of the 27th ACM international conference on information and knowledge management, Torino Italy October 22 - 26, 2018, pp. 843-852. https://doi.org/10.1145/3269206.3271761
19. Yaqing Wang, Caili Guo, Yunfei Chu, Jenq-Neng Hwang, and Chunyan Feng. "A cross-domain hierarchical recurrent model for personalized session-based recommendations." Neurocomputing 380 (2020): 271-284. https://doi.org/10.1016/j.neucom.2019.11.013
20. Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten De Rijke. "Repeatnet: A repeat aware neural recommendation machine for session-based recommendation." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, pp. 4806-4813. 2019. https://doi.org/10.1609/aaai.v33i01.33014806
21. Maditham Vikram, N. Sudhakar Reddy, and K. Madhavi. "SessionRNNRec: a deep learning based framework for modelling user sessions to generate accurate recommendations." International Journal of System Assurance Engineering and Management (2021): 1-10. https://doi.org/10.1007/s13198-021-01197-6
22. Chen Chen, Jie Guo, and Bin Song. "Dual attention transfer in session-based recommendation with multi-dimensional integration." In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 869-878. 2021. https://doi.org/10.1145/3404835.3462866
23. Tianhui Wu, Fuzhen Sun, Jiawei Dong, Zhen Wang, and Yan Li. "Context-aware session recommendation based on recurrent neural networks." Computers and Electrical Engineering, 100, (2022): 107916. https://doi.org/10.1016/j.compeleceng.2022.107916
24. Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. "A dynamic recurrent model for next basket recommendation." In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, Pisa Italy July 17 - 21, 2016, pp. 729-732. https://doi.org/10.1145/2911451.2914683
25. Yong Kiam Tan, Xinxing Xu, and Yong Liu. "Improved recurrent neural networks for session-based recommendations." In *Proceedings of the 1st workshop on deep learning for recommender systems*, Boston, USA, Sep 15, 2016, pp. 17-22. https://doi.org/10.1145/2988450.2988452
26. Dou Hu, Lingwei Wei, Wei Zhou, Xiaoyong Huai, Zhiqi Fang, and Songlin Hu. "Pen4rec: Preference evolution networks for session-based recommendation." In Knowledge Science, Engineering and Management: 14th International Conference, KSEM 2021, Tokyo, Japan, August 14–16, 2021, Proceedings, Part I, pp. 504-516. Cham: Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-82136-4_41
27. Bjørnar Vassøy, Massimiliano Ruocco, Eliezer de Souza da Silva, and Erlend Aune. "Time is of the essence: a joint hierarchical rnn and point process model for time and item predictions." In Proceedings of the twelfth ACM international conference on Web search and data mining, Melbourne, Australia, February 11 - 15, 2019, pp. 591-599. https://doi.org/10.1145/3289600.3290987
28. Sotirios P. Chatzis, Panayiotis Christodoulou, and Andreas S. Andreou. "Recurrent latent variable networks for session-based recommendation." In Proceedings of the 2nd Workshop

on Deep Learning for Recommender Systems, Como, Italy, Aug 27-31, 2017, pp. 38-45. 2017. https://doi.org/10.1145/3125486.3125493
29. Dan Li, and Qian Gao. "Session Recommendation Model Based on Context-Aware and Gated Graph Neural Networks." Computational Intelligence and Neuroscience 2021. https://doi.org/10.1155/2021/7266960
30. Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. "What to Do Next: Modeling User Behaviors by Time-LSTM." In IJCAI, Melbourne, Australia, August 19-25, 2017 vol. 17, pp. 3602-3608.
31. David Lenz, Christian Schulze, and Michael Guckert. "Real-time session-based recommendations using LSTM with neural embeddings." In Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part II 27, pp. 337-348. Springer International Publishing, 2018. https://doi.org/10.1007/978-3-030-01421-6_33
32. Yufei Feng, Fuyu Lv, Weichen Shen, Menghan Wang, Fei Sun, Yu Zhu, and Keping Yang. "Deep session interest network for click-through rate prediction." In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao China, August 10 – 16, 2019, pp. 2301-2307.
33. Serena McDonnell, Omar Nada, Muhammad Rizwan Abid, and Ehsan Amjadian. "Cyberbert: A deep dynamic-state session-based recommender system for cyber threat recognition." In 2021 IEEE Aerospace Conference (50100), pp. 1-12. IEEE, 2021. https://doi.org/10.1109/AERO50100.2021.9438286
34. Michail Salampasis, Theodosios Siomos, Alkiviadis Katsalis, Konstantinos Diamantaras, Konstantinos Christantonis, Marina Delianidi, and Iphigenia Karaveli. "Comparison of RNN and Embeddings Methods for Next-item and Last-basket Session-based Recommendations." In 2021 13th International Conference on Machine Learning and Computing, pp. 477-484. 2021. https://doi.org/10.1145/3457682.3457755
35. Dengcheng Yan, Dengcheng, Tianyi Tang, Wenxin Xie, Yiwen Zhang, and Qiang He. "Session-based social and dependency-aware software recommendation." Applied Soft Computing 118 (2022): 108463. https://doi.org/10.1016/j.asoc.2022.108463
36. Michal Dobrovolny, Jaroslav Langer, Ali Selamat, and Ondrej Krejcar. "Session Based Recommendations Using Char-Level Recurrent Neural Networks." In Advances in Computational Collective Intelligence: 13th International Conference, ICCCI 2021, Kallithea, Rhodes, Greece, September 29–October 1, 2021, Proceedings 13, pp. 30-41. Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-88113-9_3
37. Ivett Fuentes, Gonzalo Nápoles, Leticia Arco, and Koen Vanhoof. "Best Next Preference Prediction Based on LSTM and Multi-level Interactions." In Intelligent Systems and Applications: Proceedings of the 2021 Intelligent Systems Conference (IntelliSys) Volume 1, pp. 682-699. Springer International Publishing, 2022. https://doi.org/10.1007/978-3-030-82193-7_46
38. Michal Dobrovolny, Ali Selamat, and Ondrej Krejcar. "Session based recommendations using recurrent neural networks-long short-term memory." In Intelligent Information and Database Systems: 13th Asian Conference, ACIIDS 2021, Phuket, Thailand, April 7–10, 2021, Proceedings 13, pp. 53-65. Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-73280-6_5
39. Qiaolin Xia, Peng Jiang, Fei Sun, Yi Zhang, Xiaobo Wang, and Zhifang Sui. "Modeling consumer buying decision for recommendation based on multi-task deep learning." In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Torino Italy, October 22 - 26, 2018, pp. 1703-1706. https://doi.org/10.1145/3269206.3269285
40. Trinh Xuan Tuan, and Tu Minh Phuong. "3D convolutional networks for session-based recommendation with content features." In Proceedings of the eleventh ACM conference on recommender systems, Como, Italy, Aug 27-31, 2017, pp. 138-146. https://doi.org/10.1145/3109859.3109900

41. Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. "A simple convolutional generative network for next item recommendation." In Proceedings of the twelfth ACM international conference on web search and data mining, Melbourne, Australia on February 11-15, 2019, pp. 582-590. https://doi.org/10.1145/3289600.3290975
42. 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, Los Angeles, California, USA, on February 5-9, 2018, pp. 565-573. https://doi.org/10.1145/3159652.3159656
43. Wafa Shafqat, and Yung-Cheol Byun. "Enabling "Untact" Culture via Online Product Recommendations: An Optimized Graph-CNN based Approach." Applied Sciences 10, no. 16 (2020): 5445. https://doi.org/10.3390/app10165445
44. Fajie Yuan, Xiangnan He, Haochuan Jiang, Guibing Guo, Jian Xiong, Zhezhao Xu, and Yilin Xiong. "Future data helps training: Modeling future contexts for session-based recommendation." In Proceedings of The Web Conference 2020, Taipei Taiwan April 20 - 24, 2020, pp. 303-313. https://doi.org/10.1145/3366423.3380116
45. Weinan Li, Jin Gou, and Zongwen Fan. "Session-based recommendation with temporal convolutional network to balance numerical gaps." Neurocomputing 493 (2022): 166-175. https://doi.org/10.1016/j.neucom.2022.04.069
46. Rui Ye, Qing Zhang, and Hengliang Luo. "Cross-Session Aware Temporal Convolutional Network for Session-based Recommendation." In 2020 International Conference on Data Mining Workshops (ICDMW), Sorrento, Italy, November 17-20, 2020, pp. 220-226. https://doi.org/10.1109/ICDMW51313.2020.00039
47. Shahpar Yakhchi, Amin Behehsti, Seyed-mohssen Ghafari, Imran Razzak, Mehmet Orgun, and Mehdi Elahi. "A convolutional attention network for unifying general and sequential recommenders." Information Processing & Management 59, no. 1 (2022): 102755. https://doi.org/10.1016/j.ipm.2021.102755
48. Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. "A simple but hard-to-beat baseline for session-based recommendations." arXiv preprint arXiv:1808.05163 (2018).
49. Gabriel De Souza, P. Moreira, Dietmar Jannach, and Adilson Marques Da Cunha. "Contextual hybrid session-based news recommendation with recurrent neural networks." IEEE Access 7 (2019): https://doi.org/10.1109/ACCESS.2019.2954957
50. Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet A. Orgun, and Defu Lian. "A survey on session-based recommender systems." ACM Computing Surveys (CSUR) 54, no. 7 (2021): 1-38. https://doi.org/10.1145/3465401
51. Jon Atle Gulla, Lemei Zhang, Peng Liu, Özlem Özgöbek, and Xiaomeng Su. "The adressa dataset for news recommendation." In Proceedings of the international conference on web intelligence, Leipzig Germany August 23 - 26, 2017, pp. 1042-1048. https://doi.org/10.1145/3106426.3109436
52. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. "Item-based collaborative filtering recommendation algorithms." In Proceedings of the 10th international conference on World Wide Web, Hong Kong Hong Kong May 1 - 5, 2001, pp. 285-295. 2001. https://doi.org/10.1145/371920.372071
53. 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, Raleigh North Carolina, USA, April 26 - 30, 2010, pp. 811-820. https://doi.org/10.1145/1772690.1772773
54. Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian personalized ranking from implicit feedback." In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal Quebec, Canada, June 18 - 21, 2009, pp. 452-461.
55. Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. "Neural attentive session-based recommendation." In Proceedings of the 2017 ACM on Conference on

Information and Knowledge Management, Singapore, November 6 - 10, 2017, pp. 1419-1428. https://doi.org/10.1145/3132847.3132926
56. Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. "STAMP: short-term attention/memory priority model for session-based recommendation." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 1831-1839. 2018. https://doi.org/10.1145/3219819.3219950
57. Haoji Hu, Xiangnan He, Jinyang Gao, and Zhi-Li Zhang. "Modeling personalized item frequency information for next-basket recommendation." In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, China, July 25 - 30, 2020, pp. 1071-1080. https://doi.org/10.1145/3397271.3401066
58. Lizhen Wu, Chun Kong, Xiaohong Hao, and Wei Chen. "A short-term load forecasting method based on GRU-CNN hybrid neural network model." Mathematical Problems in Engineering 2020 (2020). https://doi.org/10.1155/2020/1428104
59. Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul AS Awwal, and Vijayan K. Asari. "A state-of-the-art survey on deep learning theory and architectures." Electronics 8, no. 3 (2019): 292. https://doi.org/10.3390/electronics8030292
60. Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." In International conference on machine learning, Lille France, July 6 - 11, 2015, pp. 2342-2350.
61. Mehrnaz Mirhasani, and Reza Ravanmehr. "Alleviation of Cold Start in Movie Recommendation Systems using Sentiment Analysis of Multi-Modal Social Networks." Journal of Advances in Computer Engineering and Technology 6, no. 4, (2020): 251-264.

Chapter 4
Deep Generative Session-Based Recommender System

Abstract The inherent structural sequences in sessions and the mutual influence of complex variables in different time steps make deep generative models effective solutions for a session-based recommender system (SBRS). In addition, in real-world scenarios, users usually only select a limited number of items, and their interactions in response to items are very sparse. Deep generative models that produce more training samples can help reduce the data sparsity problem. To this end, we discuss different deep generative models in SBRS in this chapter, such as autoencoders (AE), generative adversarial networks (GAN), and flow-based models (FBM).

Keywords Session-based recommender systems · SBRS · Deep generative models · AE · GAN · FBM

4.1 Introduction

The purpose of session-based recommender systems is to predict the users' subsequent transactions based on their previous short-term behaviors. This is performed when the long-term history of user behavior is unavailable or the user does not have a specific profile [1]. Previous studies in session-based recommender systems using deep learning techniques such as RNNs or CNNs have obtained more effective results than traditional sequence-based models like personalized Markov chain decomposition or feature-based matrix factorization [2].

The first research related to session-based recommender systems was presented primarily based on recurrent neural networks, which predicted the subsequent clicks of users of a session based on the hidden states they had learned so far. These methods obtain the information entropy at each time step of the observed sessions through the conditional distributions of subsequent clicks relative to previous clicks and typically choose a simple or combined parametric form. However, such a structure may not have the necessary efficiency due to the inherent structural sequences in sessions and the mutual influence of different output variables in a time step on each, considering the complex dependencies between variables in different time steps. Furthermore, click-level predictions only consider short-term reactions and ignore long-term interaction even when combined with attention-based

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2_4

mechanisms. Therefore, these approaches predict the subsequent clicks more accurately in shorter sessions but may deviate from the main goals in more extended sessions. To reduce these problems, neural networks can be strengthened by employing multimodal output distributions and uncertainty estimation [2].

On the other hand, in real-world scenarios, users usually only select a limited number of items, and their interactions in response to items are very sparse. Therefore, it is difficult to detect sequential patterns of user behavior, and it requires much more data. Meanwhile, the basic neural network-based recommender systems have many parameters whose incorrect and incomplete training may lead to challenges in optimizing complex models and, as a result, provide incorrect recommendations [3]. Therefore, using deep generative models that produce more samples for the training can help reduce the data sparsity problem [4].

There are different types of deep generative models; the widely used examples include autoregressive generative models [5], autoencoders (AE), generative adversarial networks (GAN) [6], flow-based models (FBM) [7], and energy-based models (EBM) [8]. In fact, SBRS extracts latent information related to anonymous users' interests and short-term session interactions using the potential of deep generative models such as AEs or GANs for learning meaningful data representations and embedding/reducing the dimensions of the input data [9]. A review of published research in this field shows that most of the proposed methods are based on autoencoders and generative adversarial networks. It should be mentioned that several approaches based on autoencoders have been combined with techniques such as normalizing flow, which belongs to FBM.

In this chapter of the book, we discuss the approaches utilizing deep generative models for session-based recommender systems. For this purpose, in Sect. 4.2, a brief overview of the fundamentals of these methods, the commonly used datasets, and the basic evaluation metrics used in various research are discussed. Then, in Sect. 4.3, methods based on autoencoders and then, in Sect. 4.4, approaches using generative adversarial networks are discussed and analyzed. Flow-based models in SBRS are discussed in Sect. 4.5. Section 4.6 explains the results and identifies current issues and challenges related to using deep generative methods in session-based recommender systems and provides guidelines for future research in this scope.

4.2 Fundamentals

The purpose of modeling user preferences in recommender systems is to improve the customer experience by recognizing the inherent preferences of users based on previous user behaviors. Generally, users' interests in session-based recommender systems have a complex structure. For this reason, learning methods that have a significant capacity to recognize relationships and dependencies between data should be used to learn the patterns of users' interests and recognize the similarity between the behaviors of different users based on their short-term behaviors. Deep

learning methods are practical and effective in these systems, and previous research shows the effectiveness of their application in session-based recommender systems.

Recently, more advanced deep learning approaches have been used in these types of systems to solve the mentioned challenges and optimize complex session-based recommender systems more effectively. One of the important approaches is deep generative models [1]. Generally, approaches based on deep generative models for SBRS provide recommendations by generating subsequent interactions or subsequent sessions through a carefully designed generation strategy. Deep generative models follow two objectives:

- Learning practical and accurate representations of data using unsupervised methods
- Learning joint probability distributions of data and the related classes

For example, using deep generative models to train stochastic latent variables in tasks such as natural language processing, speech generation, and machine translation has led to significant and effective results. These methods model the generation process of additional and auxiliary information, such as ratings, and generate a probabilistic latent variable framework that shares statistical strength among users and items. Different deep generative learning models provide a high capacity to learn non-linear representations of user-item interactions [2].

Deep generative models are neural networks with numerous hidden layers trained for complex estimation and high-dimensional probability distributions [10]. The most prominent goal of training this type of model is to learn intractable or unknown statistical distributions from several independent samples uniformly distributed. After successful training, deep generative models can be used to estimate the likelihood of a specific sample and generate new samples that are similar to the unknown distribution. Deep generative models are used by researchers in the field of SBRS due to their high flexibility in statistical distributions as well as their significant capacity to learn non-linear representations.

It should be mentioned that all generative models are not based on neural networks, but considering neural networks as a powerful and flexible tool, they are widely utilized to parameterize generative models and create deep generative models. Deep generative models are generally divided into four groups: autoregressive generative models, flow-based models, latent variable models, and energy-based models:

- *Autoregressive generative models*: These models use the idea of autoregressive modeling. Using the chain rule of probability, they condition their output to the data observed in the past and not to the future data. The distribution of x in the autoregressive method is calculated according to Eq. (4.1):

$$a\ p(\mathrm{x}) = p(x_0) \prod_{i=1}^{D} p(x_i | \mathrm{x}_{<i}) \tag{4.1}$$

The expression $x_{<i}$ shows all x whose index is greater than i. Modeling all conditional distributions $p(x_i|x_{<i})$ are computationally very complicated and inefficient, so they use the advantages of neural network models such as causal convolutions.

Learning long-term statistics and robust density estimators are the main features of autoregressive models. However, one of their disadvantages is that they are parameterized in an autoregressive way, and therefore, their sampling process is prolonged. Additionally, they lack a latent representation, so their internal data representations are not apparent, and thus, they are less useful for tasks such as compression or metric learning.

- *Flow-based models*: These types of models are built as a sequence of reversible transformations called normalizing flow that repeatedly replaces variables according to the change-of-variable law. Changing the formula of the variables creates a method to present the density of the random variable, performed by replacing it with an invertible transformation f according to Eq. (4.2):

$$p(x) = p(z = f(x))\left|\boldsymbol{J}_{f(x)}\right| \tag{4.2}$$

In the above equation, $\boldsymbol{J}_{f(x)}$ represents the Jacobian matrix, which can be parameterized using deep neural networks. The type of neural network should be chosen so that it can calculate the Jacobian matrix.

Flow-based models capture the accurate distribution of data and enable accurate probability assessment. Research in the field of flow-based models can be divided into two categories: models based on normalizing flows [11] and models based on autoregressive flows [12].

- *Latent variable models:* Latent variables make the latent dependencies among the observed variables to be obtained and the basic structure of the process and principles of data generation to be learned. Latent variables can provide an alternative low-dimensional representation of the observed variables. The main idea of latent variable methods is the assumption of a latent space with low dimensions, whose generating process is according to Eq. (4.3):

$$z \sim p(z), \qquad x \sim p(x|z) \tag{4.3}$$

This model assumes that the observed variable x is generated by a stochastic process based on an unobserved continuous variable z. In other words, the latent variables correspond to the hidden factors in the data, and the conditional distribution $p(x|z)$ acts as a generator. One of the most popular types of methods in this category is the probabilistic principal component analysis (pPCA), where $p(z)$ and $p(x|z)$ are based on Gaussian distribution and the dependency between z and x is linear. One of the developed examples of pPCA with an arbitrary distribution is the variational autoencoder (VAE), used to approximate the

Table 4.1 The list of research discussed using deep generative models

Deep generative models	References
Autoencoder (AE)	[2, 9, 14–24]
Generative adversarial network (GAN)	[3, 25–34]
Flow-based models (FBM)	[2]: Normalizing flows [35]: Autoregressive flows

posterior probability of $p(z|x)$ to make tractable inferences [13]. GAN networks are one of the types of latent variable models.

- *Energy-based models:* A group of generative models inspired by the laws of physics which utilize energy functions such as *E(x)*. Energy-based models (EBM), also known as non-normalized probability models, specify probability density or mass functions up to an unknown normalizing constant. The density determined using the EBM model is in the form of Eq. (4.4):

$$P(X) = \frac{1}{Z} e^{(-E(x))} \tag{4.4}$$

where the function *E(x)* (energy) is the non-linear regression function and $z = \sum_x \exp(-E(X))$ is a normalizing constant called the partition function. In other words, the distribution is defined by an exponential energy function that is further normalized to obtain values between zero and one. Energy-based models are based on the energy function, and their main idea is to formulate the energy function and estimate the partition function. One of the main groups of energy-based models is Boltzmann machines.

Generally, flow-based, autoregressive, and energy-based models and several models based on latent variables, such as variational autoencoders, can be trained sustainably, but some models based on latent variables, such as generative adversarial networks, face instability. In terms of sampling, autoregressive models have a slow process, and energy-based models are also relatively slow due to implementing the Monte Carlo method for obtaining samples, specially for objects with high dimensions. However, other methods have a fast-sampling process. Energy-based models, flow-based models, and models such as variable autoencoders can be used to learn a representation of input data and provide data with low dimensions, but autoregressive and GAN methods do not have this capability.

Due to the data sparsity and the complex structures of user interactions, many research related to session-based recommender systems have employed different deep generative models. Of course, some research has also presented a combination of several generative methods. For this purpose, AEs, GANs, and FBMs (normalizing flow and autoregressive flow) have been discussed and analyzed in this chapter.

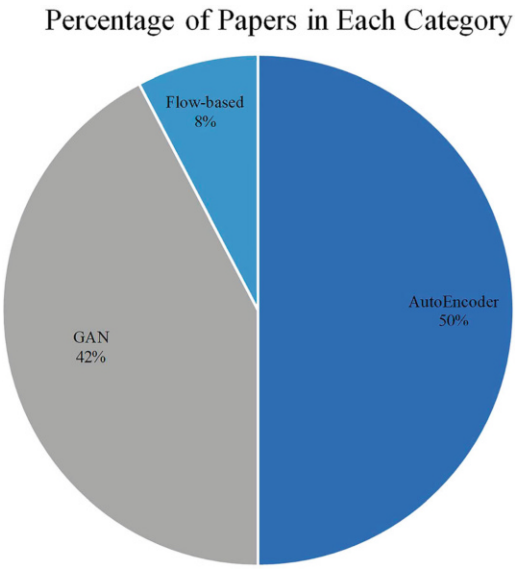

Fig. 4.1 Percentage of each type of deep generative model in SBRS

The research discussed in this chapter are shown in Table 4.1 according to the deep generative model.

The diagram in Fig. 4.1 shows the percentage of each technique used in the discussed research.

It should be mentioned that the review of research belonging to the third and fourth chapters of the book shows that discriminative models have been used more than generative models in session-based recommender systems, specially recurrent neural networks. RNNs, due to their sequential nature, have a high capacity to analyze the sequential dependencies between data in user sessions and to model users' behaviors over time. However, since deep generative methods are independent of data labels and they propose greater flexibility in this field compared to discriminative methods, researchers have been interested in using these methods more and more in session-based recommender systems.

Another noteworthy point in the review of the research of this chapter is the publication date, which shows that the approaches utilizing deep generative models are more emerging and the desire of researchers to use them in session-based recommender systems has increased in the last few years. Despite the presence of several types of deep generative models, most of the published articles use autoencoders or generative adversarial networks in session-based recommender system.

A generative adversarial network (GAN) is a generative model based on deep learning. These networks represent frameworks that build generative models based on an adversarial process. In this framework, two models are learned simultaneously:

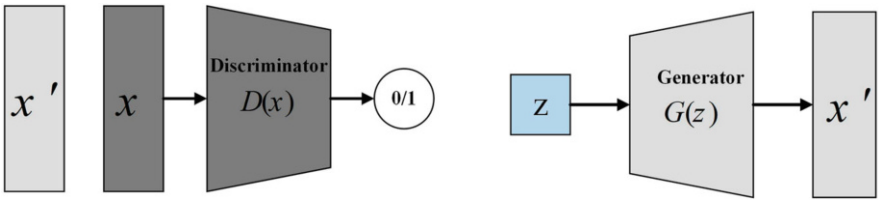

Fig. 4.2 The general architecture of a generative adversarial network

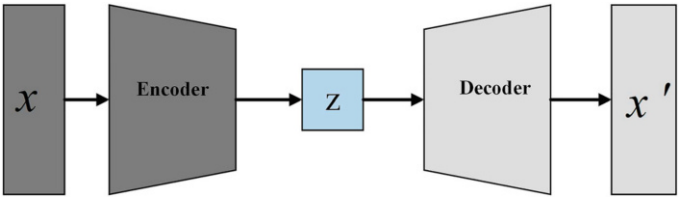

Fig. 4.3 The general architecture of an autoencoder

- A generative model that obtains the data distribution and is used to generate new acceptable examples from the problem domain. In fact, this model turns noise into fake data.
- A discriminative model that estimates the probability of a sample belonging to the training model or the model obtained from the generative method. In fact, this model is employed to classify samples as real samples (from the domain) or fake samples (made by the generative model).

The performance of this type of network corresponds to a minimax two-player game, where the goal of the training process of the generative model is to maximize the probability that the discriminative model makes a mistake. If the generative and discriminative models of GAN are defined as a multilayer perceptron (MLP) network, the whole system can be trained in the form of error backpropagation. Then, there is no need to use Markov chains or approximate inference networks [36]. Figure 4.2 shows the general architecture of a generative adversarial network.

Autoencoders are a special type of feedforward neural network in which the input and output are the same. This type of neural network was proposed by Jeffrey Hinton in the 1980s for solving unsupervised learning problems [37]. Autoencoders are trained neural networks that replicate data from the input layer to the output. As shown in Fig. 4.3, an autoencoder consists of three main parts: encoder, representation, and decoder. Autoencoders are structured to receive input and convert it into a different representation. Then, they try reconstructing the original input as accurately as possible. Autoencoders first encode the input and then reduce the size of the input to a low-dimensional representation. Finally, the autoencoder decodes the representation to generate the reconstructed data.

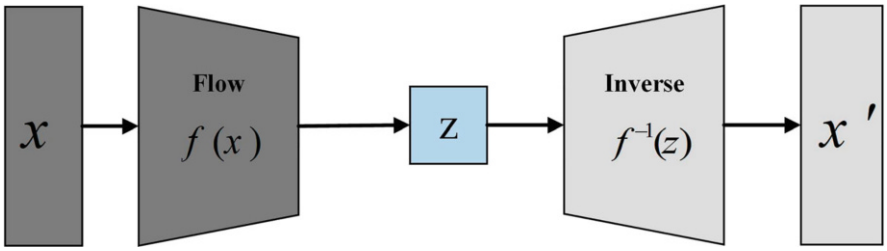

Fig. 4.4 The general architecture of a flow-based model

In recent years, various types of autoencoders have been proposed; the most widely used are sparse autoencoders (SAE), denoising autoencoders (DAE), and variational autoencoders (VAE). We found that many kinds of research related to session-based recommender systems based on deep generative models use variable autoencoders.

The distribution models used in AEs are not flexible enough to match the true posterior and the uncertainty of the recommendations. To solve the problem, improving the variational posterior distribution using the normalizing flow [38] is presented. Normalizing flow is a set of inverse transformations to the desired variables with a simple initial distribution. Compared to VAE and GAN, flow-based models have so far attracted less attention, specially in the field of session-based recommender systems, although they have unique advantages such as accurate latent variable inference and analytical likelihood evaluation. Figure 4.4 shows the general architecture of a flow-based model.

In the following two subsections, a summary of the employed datasets and evaluation metrics of the reviewed research is presented.

4.2.1 Datasets

To evaluate and validate the results of deep generative approaches in session-based recommender systems, different datasets are used from different fields. The features of these datasets have made them suitable options for evaluating the proposed approaches.

Table 4.2 shows the datasets used in different articles, including the dataset name, the domain, a brief description, and the paper that employed it.

Table 4.3 presents the information on each dataset, including the number of sessions/items/events, duration of data collection, average length of the session, type of interaction, and access link to the dataset.

The statistical information related to the dataset presented in Table 4.3 was collected from the articles or from the links that introduced them. Reviewing the evaluation section of various articles in the field of session-based recommender system shows that some datasets have been widely used to evaluate the application

Table 4.2 Widely used datasets in SBRS using deep generative models

Dataset	Domain	Description	References
Diginetica	E-commerce	The dataset includes user sessions extracted from an e-commerce search engine log	[2, 3, 16, 18]
YELP	Business	Contains users' reviews of various businesses. Each display set is simulated by collecting nine businesses with the nearest location	[27, 29]
Taobao	E-commerce	Contains the clicking and buying records of users in 22 days. We consider the buying records as positive events	[27]
Ant Financial News	News	Contains click records from 50,000 users for 1 month, involving dozens of thousands of news. On average, each display set contains five news articles	[27]
YooChoose	E-commerce	The dataset consists of 6 months of clickstreams from an e-commerce Web site	[2, 16, 18, 19, 25]
Yahoo! JAPAN's homepage	News	They sampled approximately 12 million users who had clicked at least one article from the service logs of Yahoo! JAPAN's homepage on smartphones between January and September 2016	[14]
Last.fm	Music	This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system	[19, 20, 22, 27]
RecSys Challenge 2015	E-commerce	This dataset comprises clickstream data to user sessions with an e-commerce Web site	[2, 16, 25, 27]
Studo	Job	Studo is a proprietary dataset collected from the online platform Studo Jobs, a job-seeking service for university students	[9]
RecSys17	Job	The RecSys17 is the latest version of the data provided by XING after the RecSys Challenge 2017	[9, 22]
CareerBuilder12	Job	It is from an open Kaggle competition, called Job Recommendation Challenge, provided by the online employment Web site CareerBuilder	[9]
Amazon	E-commerce	Amazon is an e-commerce dataset, where [3] focuses on the baby, beauty, and cellphone domains	[3, 31]
Netflix	Video	Netflix is a well-known dataset recording the ratings of users on a catalog of movies. Because the timestamp of each rating is available, it is possible to construct for each user the sequence of movies he rates	[17, 28, 30]
Booking.com	Travel	Booking.com recently organized the WSDM WebTour 2021 Challenge. The dataset consists of over a million anonymized hotel reservations based on real data. The challenge's goal is to recommend the final city of each trip	[15]

(continued)

Table 4.2 (continued)

Dataset	Domain	Description	References
MovieLens	Movie	It consists of users' sequential rating records for different categories of movies on the MovieLens site	[3, 17, 20, 21, 27–29, 31, 35]
CiteULike	Research paper	In the CiteULike dataset, one user annotating one research paper at a certain time may have several records to distinguish different tags	[35]
Retailrocket	E-commerce	The data has been collected from a real-world e-commerce Web site. It is raw data, i.e., without any content preprocessing; however, all values are hashed due to confidential issues	[22]

of different approaches based on discriminative, generative, or hybrid methods. For example, YooChoose, MovieLens, RecSys Challenge 2015, and Diginetica datasets have been used in many articles. Of course, each proposed approach, according to the nature and type of their performance, has considered different preprocessing operations for the dataset. For example, in [25] for the YooChoose dataset, only sessions with more than five interactions are considered, or in [3] for the Diginetica dataset, only users or items are considered that have participated in more than five interactions. Sessions with more than two items have been considered in [16] for the Diginetica and YooChoose datasets. In some research, different modifications have been made to the original data to match it with the proposed method, for example, in [17], the data related to the scores of the Netflix and MovieLens datasets have been converted to binary values.

4.2.2 Evaluation

To accurately evaluate and analyze the results of new approaches to session-based recommender systems, several previous approaches in this field are usually used as a baseline. Some of these methods utilize the basic methods of session-based recommender system. Some others are based on neural networks and specific to the evaluation of deep learning session-based recommender systems. On the other hand, one or more evaluation metrics are used to assess the performance of deep generative models in the proposed approaches. Each of these evaluation metrics considers the proposed methods from a specific perspective. In these subsections, first, widely used baseline methods are introduced, and then the relevant metrics are discussed and reviewed.

The most widely used baselines are as follows. It should be noted that some of them have been introduced as baselines in the previous chapter. However, to keep the comprehensiveness and consistency of the content of this chapter, they are also repeated in this section:

Table 4.3 Characteristics of widely used datasets

Domain	Dataset	Number of sessions	Number of items	Number of events	Timespan in days	Average session length	Interaction type	Access link
E-commerce	YooChoose 1/64	425,757	16,766	557,248	182	6.16	Click/buy	https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015
	Diginetica	204,789	43,136	993,483	–	5.12	Click	https://competitions.codalab.org/competitions/11161#learn_the_details-data2
	RecSys Challenge 2015	7,981,581	37,486	31,708,505	182	3.97	Click/buy	https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015
	Taobao	186,857	14,746	3,214,637	22	17.2	Click	https://tianchi.aliyun.com/dataset/649
	Amazon (baby, beauty, cellphone)	–	29,583	553,733	19 years	–	Reviews and metadata	http://jmcauley.ucsd.edu/data/amazon/
Video	Netflix	480,189	17,770	100,480,507	6 years	–	Rating (1–5)	https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data/discussion
	MovieLens-1M	6040	3706	1,000,209	17 years	–	Rating (1–5)	http://files.grouplens.org/datasets/movielens/
	Yahoo! JAPAN's homepage	166 million	2 million	1 billion	14	–	Click	https://webscope.sandbox.yahoo.com/catalog.php?datatype=l
POI	Gowalla	830,893	29,510	1,122,788	240	3.85	Check-in	https://snap.stanford.edu/data/loc-gowalla.html
Music	Last.fm	169,576	449,037	288,7349	95	17.03	Click	http://millionsongdataset.com/lastfm/
Job	Studo	26,875	1111	191,259	90	6.98	View, apply, share, detail	Not public
	RecSys17	16,322	15,686	55,380	90	3.62		

(continued)

Table 4.3 (continued)

Domain	Dataset	Number of sessions	Number of items	Number of events	Timespan in days	Average session length	Interaction type	Access link
							Click, mark, and apply	http://www.recsyschallenge.com/2017/
	CareerBuilder12	120,147	197,590	661,910	90	5.64	Apply	https://www.kaggle.com/c/job-recommendation
Travel	Booking.com	217,686	39,901	1,166,835	1 year	5.36	Reservation	https://github.com/bookingcom/ml-dataset-mdt

POP More popular items are always recommended. The POP is effective and straightforward simultaneously and is often a strong baseline in specific domains [39].

S-POP The most popular items in the current session are recommended. The recommendation list changes based on the number of events that are related to particular items. This baseline is useful for the domains with high repetitiveness [39].

Item-KNN Items similar to the actual item are recommended, and the similarity between them is measured based on the cosine similarity measure of their session vectors. In other words, it is the number of co-occurrences of two items in sessions divided by the square root of the product of the number of sessions in which the individual items occurred. This method is very effective for evaluating item-to-item recommendation methods [39].

FPMC A hybrid model for the next-basket recommendation based on the factorizing personalized Markov chains. This method is adapted in [2] for the scenario of session-based recommender systems. In fact, the user's latent representations are omitted when calculating the scores of the recommendations.

BPR-MF It utilizes matrix factorization which is optimized for pairwise ranking objective functions through stochastic gradient descent. Methods based on matrix factorization cannot be used in session-based models because there is no pre-computed feature vector for new sessions. This problem is overcome by using the average vectors of the items that belong to each session [49].

GRU4Rec A technique based on recurrent neural networks, which is one of the first approaches to using deep learning techniques in session-based recommender system. This method is based on GRU and is used to overcome the problem of gradient vanishing [39].

GRU4Rec+ This method is an improved version of GRU4Rec that uses data augmentation and considers shifts in the distribution of input data to improve the performance of GRU4Rec [2].

GRU4Rec++ This method is one of the most recent methods that extend GRU4Rec by introducing an improved sampling strategy pattern [2].

NARM An improved version of GRU4Rec, which performs session modeling by introducing a hybrid encoder based on the attention mechanism. In this technique, global and local encoders are defined, the global encoder corresponds to the GRU4Rec method, and the local encoder is proposed for adding the attention mechanism to the model, respectively [18].

STAMP This method is based on a Short-Term Attention/Memory Priority Model and, unlike the NARM method, is not based on a recurrent neural network. In this method, users' general interests are obtained through the long-term memory data of the session context, and their short-term interests are also recognized through short-term memory [40].

ReLaVaR A Bayesian version of GRU4Rec, which considers the recurrent units of the network as stochastic latent variables with some prior distributions and infers the corresponding posterior probabilities for prediction and recommendation. This method provides a session-based recommender system that works based on variational inference at the item level and uses the independent Gaussian distribution as the prior probability of the items [2].

VRM A variational autoencoder approach for session-based recommender systems. Unlike ReLaVaR, which is a variational method at the item level, the VRM model performs stochastic inference on the session level [23].

CDAE A well-known model to recommend top N recommendations, which uses denoising autoencoders and learns the data model from corrupted inputs [41].

CASR This counterfactual data augmentation method has been proposed for sequential recommender systems [42]. It is composed of a sampler model and an anchor model. The sampler model generates counterfactual sequences from the real ones. The anchor model provides the final recommendation list and is trained based on both real and counterfactual sequences.

SASRec The first model of the session-based recommender system based on the self-attentive mechanism [43].

In this section, a number of evaluation metrics that are used more in this field have been discussed in the following:

- *Recall:* This metric is calculated based on the number of relevant items that are among the top N items in the recommendation list, and the rank of the relevant items in the N list is unimportant, and it is calculated using Eq. (4.5):

$$\text{Recall@}N = \frac{\text{Number of relevant items in top } N \text{ list}}{\text{Total of relevant items}} \tag{4.5}$$

- *Mean Reciprocal Rank (MRR):* MRR focuses on the rank of relevant items in the list of recommendations. It shows that placing a relevant item at the top of the recommendation list significantly impacts user satisfaction and is calculated using Eq. (4.6):

$$\text{MRR@}N = \frac{1}{Q}\sum_{i=1}^{Q}\begin{cases}\frac{1}{\text{rank}_i} & \text{if } \text{rank}_i \le N \\ 0 & \text{otherwise}\end{cases} \tag{4.6}$$

  where Q is a sample of recommendation lists and rank_i refers to the rank position of the relevant item for the i-th recommendation list.

- *Precision @ N:* This metric evaluates the number of relevant items relative to the total N items recommended in the list, and it is calculated using Eq. (4.7):

$$\text{Precision@}N = \frac{\text{Number of relevant items in top } N \text{ list}}{\text{Total of } N \text{ items}} \tag{4.7}$$

- *Coverage@N:* It checks the coverage of the items. Item coverage measures the percentage of items that are ever recommended, and the variety of the recommended items in the recommendation list is considered. Its goal is to recommend a high percentage of various items to the user. This metric is calculated using Eq. (4.8):

$$\text{Coverage@}N = \frac{\text{Distinct items that appeared in any top} - N \text{ recommendation}}{\text{All distinct recommendable items}} \tag{4.8}$$

- *$nDCG_p$:* This metric is based on cumulative gain (CG). The cumulative gain is the sum of the graded relevance values of all items in a recommendation list. nDCG is computed as the ratio between discounted cumulative gain (DCG) and idealized discounted cumulative gain (IDCG). Equations (4.9), (4.10), and (4.11) show how to calculate this measure.

$$\text{DCG}p = \sum_{i=1}^{p} \left(\frac{2^{r_i} - 1}{\log_2(i+1)} \right) \tag{4.9}$$

$$\text{IDCG}p = \sum_{i=1}^{\text{REL}p} \left(\frac{r_i}{\log_2(i+1)} \right) \tag{4.10}$$

$$n\text{DCG}p = \frac{\text{DCG}p}{\text{IDCG}p} \tag{4.11}$$

In the above equations, r_i is the graded relevance of the result at position i, and RELp represents the list of relevant items (ordered by their relevance) up to position p.

- *MAP:* This metric calculates the average precision. In fact, after each relevant item is recommended, the precision is measured, and the average is calculated using Eq. (4.12):

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{Ave}P(q)}{Q} \tag{4.12}$$

In this relation, *P(q)* is the precision of query *q,* and parameter Q is the number of queries.

- *Hit Rate@N:* It is the percentage of times in which relevant items are retrieved among the top N ranked items, and it is calculated using Eq. (4.13):

$$\text{Hit Rate@}N = \frac{1}{Q}\sum_{i=1}^{Q}\begin{cases}1 & \text{if } \text{rank}_i \leq N \\ 0 & \text{otherwise}\end{cases} \tag{4.13}$$

  where Q is a sample of recommendation lists and rank_i refers to the rank position of the relevant item for the i-th recommendation list.

- *Mean Absolute Error (MAE):* This metric is one of the most common errors of prediction factors, which calculates the mean absolute value of the difference between the score predicted by the system and the actual score of the item. The mean absolute error indicates the degree of closeness of the recommendations to reality. This measure can be calculated from Eq. (4.10).

$$\text{MAE} = \left(\frac{1}{N}\sum_{i\in O_u}|P_{u,i} - r_{u,i}|\right) \tag{4.14}$$

- *Root Mean Square Error (RMSE):* The metric of the root mean square error of the predicted rank is more effective than the mean absolute error in problems where the errors are more considerable, and it is calculated using Eq. (4.14):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i\in O_u}(P_{u,i} - r_{u,i})^2} \tag{4.15}$$

  In Eqs. (4.14) and (4.15), $P_{u,i}$ is the predicted score for the item i by user u, $r_{u,i}$ is the actual value of the score assigned to item i by user u, O_u is the set of items rated by user u, and N is the total number of predictions made by the system.

- *Area Under the ROC Curve (AUC):* Another important metric used to determine the efficiency of recommender systems is the AUC. The larger the AUC value, the more favorable the final system performance is evaluated. The ROC (receiver operating characteristic) space is formed by two indices FPR on the horizontal axis and TPR on the vertical axis, as calculated by Eqs. (4.16) and (4.17), respectively. The line that connects two points (0,0) and (1,1) divides the ROC space into two parts. The area above this line is the favorable area and below the line is the unfavorable area. Therefore, the AUC is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{4.16}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{4.17}$$

The maximum value of this metric is equivalent to one and occurs in a situation where the recommender system is ideal and can recognize all positive samples. The AUC measure, unlike other measures for deciding the efficiency of classification methods, is independent of the classification threshold. Therefore, this measure indicates the output reliability of the system.

- *Expected Popularity Complement (EPC)*: This metric shows the ability of the recommender system to introduce items that have not been recommended in the system before. This measure is calculated based on Eq. (4.18):

$$\mathrm{EPC@}k = \frac{1}{|S|}\sum_{s\in S}\frac{1}{|R_k|}\sum_{R_i\in R_k}\mathrm{disc}(i)p(\mathrm{rel}|R_i,s)(1-p(\mathrm{seen}|R_i)) \tag{4.18}$$

In this regard, $\mathrm{disc}(i)$ is the discount factor for the weight of recommendation rank i, and $p(\mathrm{seen}|R_i)$ is the probability that the i-th recommended item was already seen in the system.

- *Expected Profile Distance (EPD)*: Unlike the EPC measure, the EPD uses the semantic content of recommendations and shows how surprising and unexpected recommendations are for a specific session history. This measure is calculated based on Eq. (4.19):

$$\mathrm{EPD@}k = \frac{1}{|S|}\sum_{s\in S}\frac{1}{|R_k||H_s|}\sum_{R_i\in R_k}\sum_{H_j\in H_s}\mathrm{disc}(i)p(\mathrm{rel}|R_i,s)d\left(R_i|H_j\right) \tag{4.19}$$

In the above equation, $\mathrm{disc}(i)$ is the discount factor for the weight of recommendation rank i, and $d(R_i|H_j)$ represents the dissimilarity between R_i and H_j. The session-based novelty is given as the average dissimilarity of all items in the list of recommended items R_i and items in the current session H_j.

Table 4.4 shows the different evaluation metrics used in different articles on session-based recommender systems using deep generative models.

Table 4.4 Widely used evaluation metrics in SBRS using deep generative models

Evaluation metrics	References
Mean reciprocal rank (MRR)	[2, 3, 9, 14, 16, 18, 19, 21, 22, 28–30]
Recall@n	[2, 3, 17–19, 22, 24, 31, 35]
nDCG	[3, 9, 14, 17, 20, 21, 26, 28–31, 35]
AUC	[14]
Precision@n	[16, 17, 25, 27, 28, 35]
Accuracy	[15]
RMSE	[25]
MAP	[26, 27, 30, 35]
EPC	[9]
EPD	[9]
Coverage	[9]
Hit Rate@n	[20, 21]

4.3 Session-Based Recommender System Using Autoencoder

Before looking at the approaches of autoencoder models in session-based recommender system, an overview of AE and the reasons that made it an appropriate choice for SBRS are provided.

4.3.1 Why Autoencoder?

With the advancement of neural networks and the increase in the computation power of computer systems, the use of deep generative models has become one of the most widely used approaches in different fields of artificial intelligence. In recommender systems, deep generative models have been widely used, and one of the most popular models in this field is autoencoders. The autoencoders are utilized to learn meaningful representations of data such as embedding, reducing input data dimensions, data reconstruction, etc. [44]. Moreover, the autoencoders are used in recommender systems to compress, cluster, and reduce data dimensions to recognize latent similarities between items or users and predict users' interests based on them. Recommender systems that use autoencoders have a more effective performance in terms of noise management, as well as the use of multimedia data resources, compared to traditional recommender systems. The results of the evaluation show that autoencoder-based recommender systems could generate more accurate results.

Recently, research related to session-based recommender systems has also used autoencoders for extracting hidden information related to the interests of anonymous users in short sessions. In the proposed approaches, autoencoders are employed to embed and generate semantic representations of users, items, or sessions, and

simultaneously, while maintaining the most important features, they also reduce the dimensions of the data. Indeed, autoencoders have changed the architectures of session-based recommender systems, and by analyzing and reconstructing users' experiences, they have provided more opportunities to increase their satisfaction with the system's performance. Autoencoders, alone or combined with other deep learning methods, try to reduce the weaknesses of recommender systems, such as data privacy, and effectively learn non-linear relationships between users and items. Data privacy can be alleviated by learning knowledge from different data sources, including contextual, textual, and image data.

Autoencoders are categorized under unsupervised learning. The structure of the autoencoder is divided into two parts: encoding and decoding. In encoding, the input data is mapped to the feature space, and in decoding, it is converted back to its original form from the feature space. The main part of an autoencoder is the hidden layer used as the extracted feature for classification. The autoencoder receives a set of data and, by encoding them, tries to represent the inputs. The autoencoder is trained so that the weights produced in the layers make the output have the minimum possible deviations from the input or equal in an ideal case. When training an autoencoder, the model parameters such as the number of middle layer nodes, the cost function, and the number of layers affect the performance of the model and should be set in advance.

Figure 4.5 shows the general structure of an autoencoder, which consists of three layers: input layer, hidden layer, and output layer.

The number of neurons in each input, hidden, and output layer is equal to n, m, and n, respectively. The input layer and the hidden layer construct the encoder, and the hidden layer, together with the output layer, constructs the decoder. As shown in Fig. 4.4, the encoder converts $x = \{x_1, x_2, \ldots, x_n\}$, which is a high-dimensional input data, into $h = \{h_1, h_2, \ldots, h_m\}$, which is a low-dimensional latent representation. This transformation is performed through the f function in Eq. (4.20):

$$h = f(x) = s_f(Wx + b) \tag{4.20}$$

In the above equation, s_f is the activation function. The encoder parameters include a weight matrix W with dimensions $m \times n$ and a bias vector $b \epsilon R^m$. The decoder reconstructs the representations of the hidden layer h and reaches the data $\acute{x} = \{\acute{x}_1, \acute{x}_2, \ldots, \acute{x}_n\}$ using the g function in Eq. (4.21):

$$\acute{x} = g(h) = s_g\left(\acute{W}h + \acute{b}\right) \tag{4.21}$$

In the above equation, s_g is the activation function. The decoder parameters include a weight matrix $W^{'}$ with dimensions $n \times m$ and a bias vector $\acute{b} \epsilon R^n$. s_g and s_f functions are usually non-linear activation functions such as hyperbolic tangent function, sigmoid, etc. These non-linear activation functions help the autoencoder to learn more important and useful features than the PCA method by minimizing the reconstruction error between x and $\acute{x}$ and obtaining a d-dimensional representation of

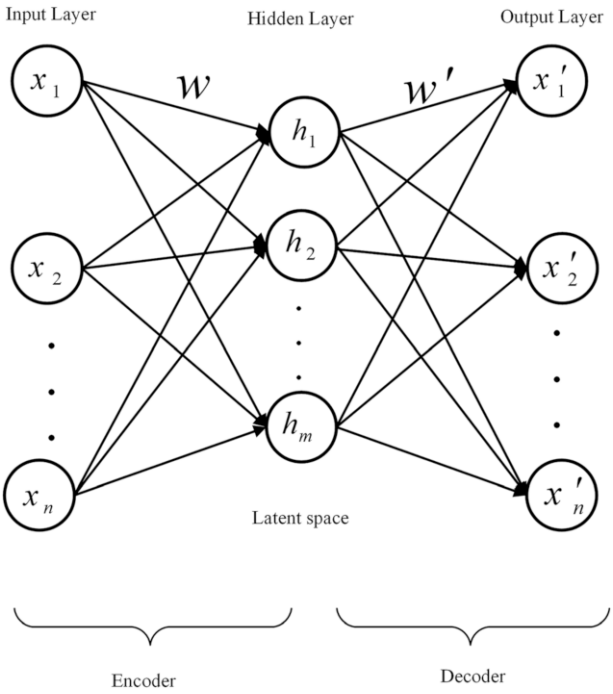

Fig. 4.5 The layers of the autoencoder

the input data. There are two methods of squared error and cross-entropy to formulate the reconstruction error, which is calculated based on Eqs. (4.22) and (4.23), respectively:

$$E_{AE}(x, \acute{x}) = \|x - \acute{x}\|^2 \tag{4.22}$$

$$E_{AE}(x, \acute{x}) = -\sum_{i=1}^{n} (x_i \ \log \acute{x}_i + (1 - x_i) \log(1 - \acute{x}_i)) \tag{4.23}$$

A regularized term can be added to calculate the reconstruction error and make the loss function of the autoencoder. The loss function can be optimized through methods such as SGD (stochastic gradient descent). Although this model is often efficient, it can become very ineffective if errors occur in the first layers. A proper technique to eliminate this problem is pre-training the network with initial weights that approximate the final solution.

Solutions that are presented based on autoencoders may face challenges that affect the usability and robustness of the model. These challenges include [45]:

- Weight initialization: With large initial weights, autoencoders usually detect weak local minima, and if the initial weights are small, the gradients of the initial layers are small, making it impossible to train autoencoders with many hidden layers. Random selection of initial values also affects the results.
- Model configuration: The model configuration, including the number of layers and their width, makes the network look for a specific image of the data while preserving relevant details.
- Hyperparameters: There are several hyperparameters for autoencoders that are difficult to set. These hyperparameters are learning rate, weight cost, dropout function, batch size, number of epochs, number of layers, number of nodes of encoding/decoding layers, a type of activation function, initialization of weights, and optimization algorithm.

If the neural network constructing an autoencoder is a deep network, it is called a deep autoencoder. In this architecture, the number of hidden network layers is more than one. Currently, very deep networks can be easily trained by GPUs. Recently, different types of autoencoders have been proposed in various research on session-based recommender systems, which are briefly reviewed as follows:

Denoising Autoencoder The autoencoder sometimes adapts only to the input data instead of finding the most salient feature (this is an example of overfitting). The denoising autoencoder adds a little noise to the input cell. By doing this, the encoder is forced to reconstruct the output from a corrupted input and acquire more robust features. The input in this network is a corrupted version $\tilde{x} \in R^n$ of the original input $x \in R^n$. This autoencoder does not simply copy the input to the output but denoises the data and then produces the input from the noisy version. The loss function minimizes the error in the noisy input. The general form of this type of autoencoder is shown in Fig. 4.6.

Convolutional Autoencoder The convolutional autoencoder is a convolutional neural network used as an advanced method in unsupervised learning based on convolutional filters. In a convolutional autoencoder, the model can learn optimal filters that minimize the reconstruction error instead of manually engineering convolutional filters. Once these filters are learned, they can be applied to any input to extract features. Therefore, these features can be used to do anything requiring a compact input representation, such as classification. A convolutional autoencoder learns to encode the input into a set of simple signals and then reconstructs the input from them. In this type of autoencoder, the encoder layers are called convolutional and the decoder layers are called deconvolution layers. The general form of this type of autoencoder is shown in Fig. 4.7.

Variational Autoencoder Compared with the autoencoder, the variational autoencoder compresses probabilities instead of features. Despite the small differences between the two mentioned neural networks, each of them answers a different

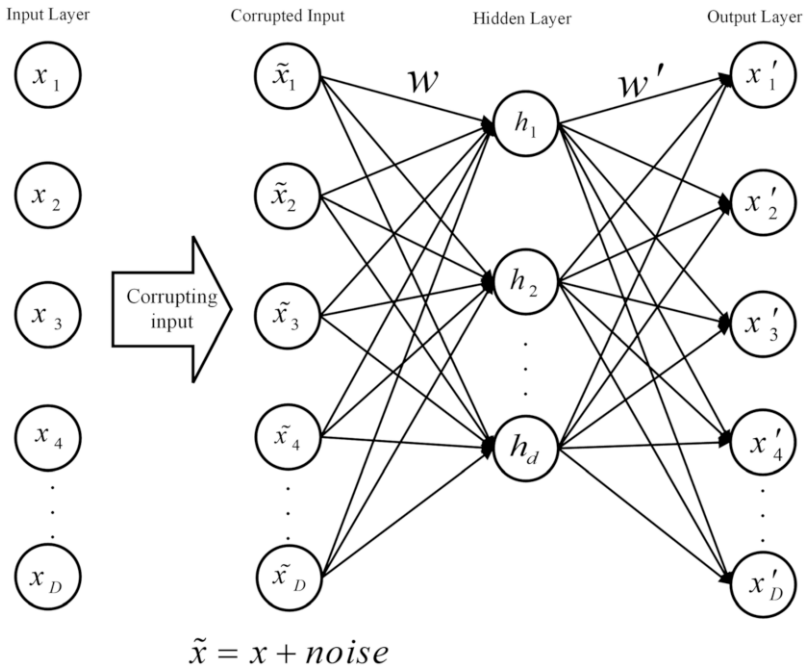

Fig. 4.6 The general architecture of denoising autoencoder

Fig. 4.7 The general architecture of convolutional autoencoder

question. Autoencoder answers the question "How can the data be generalized?". In contrast, the variational autoencoder answers "How strong is the connection between two events? Should the fault be distributed between the two events, or are they completely independent?". The variational autoencoder is an example of a deep latent variable model that uses neural networks to approximate the posterior inference of latent variables and generate data samples. Indeed, a variational autoencoder is a probabilistic generative model in which the probability density *p(x)* is modeled through a latent variable *z*. Its goal is to model *p(x)* so that sampling the distribution leads to the generation of ideal samples from the input dataset that does not exist in the original form. In fact, the variational autoencoder can generate new data samples similar to the data samples the model has seen during the training process. The general form of this type of autoencoder is shown in Fig. 4.8.

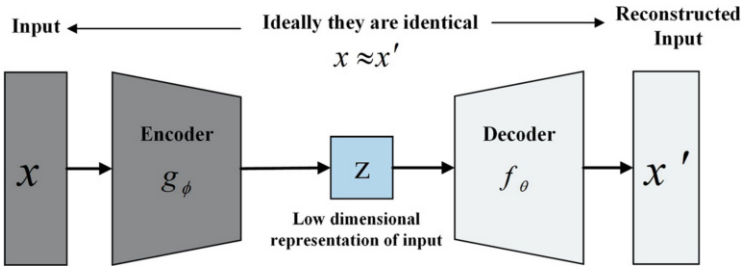

Fig. 4.8 The general architecture of variational autoencoder

4.3.2 Autoencoder Approaches

Autoencoders are one of the most powerful methods for extracting the main features of data. These types of neural networks are used for unsupervised learning with the aim of dimensionality reduction, optimal embeddings, and generative modeling. To use the advantages of autoencoders and overcome the problems of previous session-based recommender systems, different types of these systems have utilized autoencoders. Each research discussed in this section has used different types of autoencoders for their proposed model.

One of the first research that used autoencoders in session-based recommender system was proposed by Li et al. [18]. The main idea is to create a latent representation of the user's current session and make predictions. This method includes an attention-based mechanism that works along with an encoder to model users' sequential behaviors, determine the user's main goal in the current session, and finally create a unified representation of the session. Figure 4.9 shows the schematic of this method, which is called neural attentive recommendation machine (NARM).

As shown in the figure, the encoder converts a sequence of input clicks $x = \{x_1, x_2, \ldots, x_{t-1}, x_t\}$ into a set of high-dimensional arrays $h = \{h_1, h_2, \ldots, h_{t-1}, h_t\}$ and sends it along with the attention signal at time $t(\alpha_t)$ to the generator of session features to create a representation of the current session and decoding at time $t(c_t)$. Finally, c_t is transformed into an activate function using the U matrix to create a ranking list over all items. Following the successful use of GRUs in the GRU4Rec method, this method also uses GRUs to model user sessions. In the proposed architecture, a global encoder is considered, which is based on GRU4Rec and is used to model sequential user behaviors. A local encoder is also used to detect the main purpose of the user in the current session.

While in the previous methods, only the last hidden state was used to encode the session, NARM uses all the hidden states of the GRU to encode the sessions. The similarity between the last global hidden state and all previous global hidden states is calculated, and using this similarity, the final local encoding of the session is determined through the sum of all hidden states that are weighted based on their similarity.

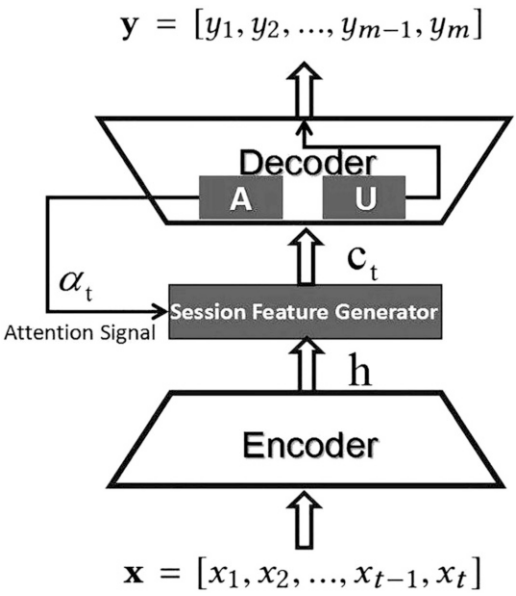

Fig. 4.9 The schematic of NARM [18]

The main disadvantage of NARM is that a session may contain noise or a collection of choices from multiple users, so not all dependencies may be correctly detected and considered. Many session-based recommender systems only focus on the user's current session and ignore the collaboration information of previous sessions. The information from the previous sessions provides the behavior of other users who may have similar interests to the intended user. To solve this problem, a Collaborative Session-based Recommendation Machine (CSRM) with parallel memory modules has been proposed by Wang et al., which uses a memory network to encode the current session of the user [19]. In addition to the information from the current session, this method also uses the information from neighbor sessions to provide recommendations.

Figure 4.10 shows the architecture of CSRM. In CSRM, two modules based on neural networks are implemented in parallel: the inner memory encoder (similar architecture to the NARM) and the outer memory encoder. The inner memory encoder is made of two submodules, one of which obtains the user's global behavior based on the order of his interactions and the other is used to pay attention to the user's specific behavior which is reflected by relatively important items in the current session. Their outputs are linearly combined and determine the main goal of the user and his interests in the session. On the other hand, the external memory encoder implements a session-based collaborative refinement approach to extract knowledge from other similar sessions. Finally, the information of both encoders is combined through a fusion gating mechanism.

In addition to the CSRM, Liang et al. proposed a session-based recommender system with a dual attention-based neural network that uses a hybrid encoder to solve the problem of ignoring the user's interest in previous sessions [22]. The

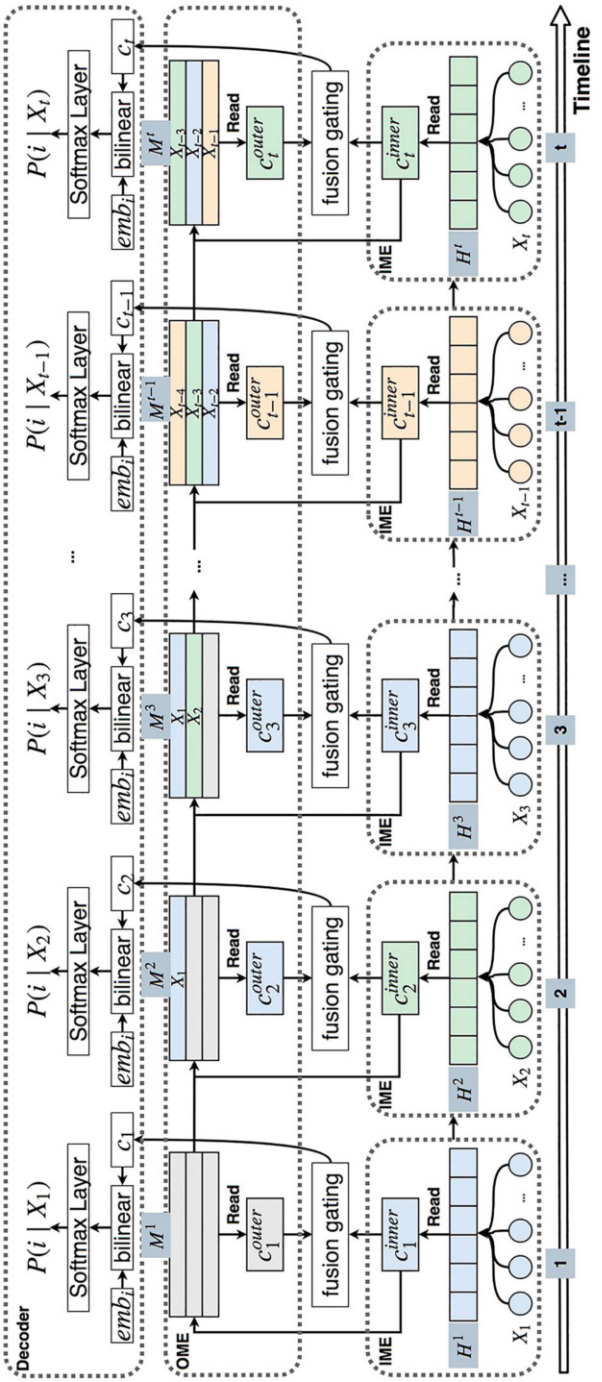

Fig. 4.10 The architecture of the CSRM [19]

hybrid encoder consists of the session encoder and the user encoder. The session encoder, using a session-level attention mechanism, examines the user's interests and goals in the current session, and the user encoder, using a user-level attention mechanism, distinguishes the user's interests between different sessions. Both encoders have employed GRU. Finally, using a decoder, the scores of the proposed items are calculated.

Santana et al. applied some modifications to the NARM and proposed a travel recommendation system that was the subject of the WSDM WebTour 2021 Challenge [15]. In this method, categories and dense features of users, cities, and trips are combined with data about the history of trips. The proposed architecture of this method is shown in Fig. 4.11, the core of which is the NARM module, which works the same as the process described above, but the size of the inputs, the bottleneck of the latent representation, and the outputs have been changed.

Statistical user features use an autoencoder trained on the same training data to embed user information. In addition to providing a dense representation of user features, the autoencoder ensures that similar users are close to each other in the vector space. At the beginning of the work, the input features are concatenated and take two paths. The first group of features passes through an attention-based layer before being sent to the NARM module. This is an essential step in relating different positions of the same input sequence. In the second path, the features bypass the feature bottleneck created by the NARM module, which results in improved decoded performance and provides more contextual information for the session.

Okura et al. proposed a news recommender system based on the embedding technique that uses a denoising autoencoder [14]. This is a three-step approach that creates distributed representations of news articles using denoising autoencoders in the first step. Then, based on the order and history of users' searches, it creates representations of users employing recurrent neural networks. Finally, for each user, the inner product between the article-user representations is performed for the degree of relationship between them. This production between article-article representations is also performed to avoid the repetition of similar information in different articles. Recurrent neural networks are used to learn user characteristics based on the previous data of his searches.

In this method, denoising autoencoders with weak supervision are used to represent news articles based on news text. Since the hidden layer contains input data information, the hidden layer (h vector) is usually used to represent the input. Based on this feature and taking into account that the greater the similarity between two input articles, the larger the inner product of the two vectors presented as ${h_0}^T h_1$ becomes, this method uses a triple set (x_0, x_1, x_2) of articles as an autoencoder input. x_0, x_1 are the articles with the same category, and x_0, x_2 belong to a different category (x is the original input vector). A penalty function is considered to detect the similarity of the articles based on the classification using the inner product of the input vectors. In this method, instead of using the stochastic corruption of the input data, constant decay is used, which is based on the corruption rate in the training phase (p is the corruption rate). As a result, the hidden vector will be unique at the time of use for each article. Multiplying $1-p$ is effective in equalizing the input

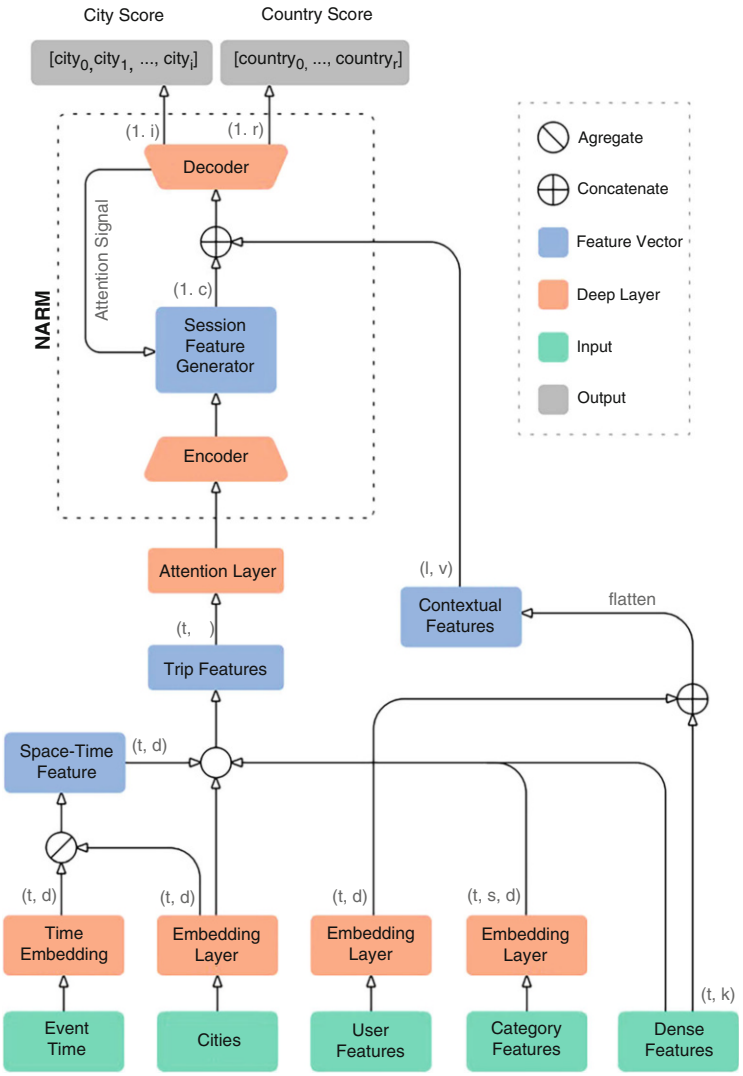

Fig. 4.11 The architecture of the adapted NARM [15]

distribution of middle-layer neurons in the learning phase with masking noise and without applying this noise.

An approach based on the combination of deep learning and latent variable modeling has been presented by Sachdeva et al. for a sequential recommender system called SVAE [17]. The authors assume that at a certain time, the choice of a certain item is affected by latent factors that model the interests of the user. It is worth mentioning that latent factors are influenced by the short- and long-term history of user interests and interactions. The previously proposed methods showed

that recurrent neural networks could consider short- and long-term dependencies and could be used in a dynamic environment to obtain the optimum results.

SVAE utilizes a recurrent neural network architecture with different levels of abstraction to capture latent temporal dependencies and user preferences. SVAE uses sequential variational autoencoders that use variational autoencoders to model user preferences along with latent variables and temporal dependencies. SVAE models the latent dependencies using a recurrent neural network before sending the latent dependencies for prediction to variational autoencoders.

Another framework based on gap-filling using masked convolutional operators has been proposed by Yuan et al., which allows simultaneously considering future and past context data without data leakage [21]. In GRec (Gap-filling-based Recommender) approach, the encoder takes the sequence of sessions relatively complete as an input, and the decoder predicts the masked items based on the output of the encoder and its embeddings. Therefore, in GRec, the encoder must be aware of the user's interests and preferences through the user's unmasked operations, and simultaneously, the decoder must predict the next item based on the previous context and the encoded general interests of the user.

GRec uses convolutional neural networks with sparse kernels, which has two main advantages: (1) creating an autoregressive mechanism to construct the sequence and (2) creating two-side contexts for encoding. The projector neural network proposed in GRec increases the representational bandwidth between the encoder and the decoder. The encoder is implemented with a set of one-dimensional dilated convolutional neural networks, in which every two dilated layers by a residual block are wrapped to prevent the gradient vanishing of both dilated layers. The decoder also consists of embedding layers, the projector, and causal convolutional neural networks, which each position can only attend to the left.

According to different types of autoencoders, Lacic et al. have developed a session-based job recommender system using different autoencoders [9]. The autoencoder is used to encode user sessions, and it is trained based on different job datasets. These datasets include user interactions that are extracted from sessions and content features of job postings where interactions occur during a session. In this research, three types of autoencoders, including classic autoencoder, denoising autoencoder, and variational autoencoder, have been investigated to represent a session. The input data of the three autoencoders are a binary representation of the interactions of each session, and the dimensions of the input and output layers are the same. In the classic autoencoder, which is the simplest type, there is a hidden layer between the input and output layers representing the session and providing a latent representation of the session using a mapping function. In the denoising autoencoder, additive Gaussian noise is used to corrupt the input, which is applied to the input layer with a probability of 0.5. The variational autoencoder uses variational inference to extract the latent representation of the session and estimate the intractable posterior distribution with a simpler variational distribution. Evaluations show using a variational autoencoder achieves better results. Figure 4.12 shows how to model sessions using an autoencoder. AE_{Int}, which is a standard autoencoder, considers user's interaction data, and to combine this with job content data, AE_{Comb}

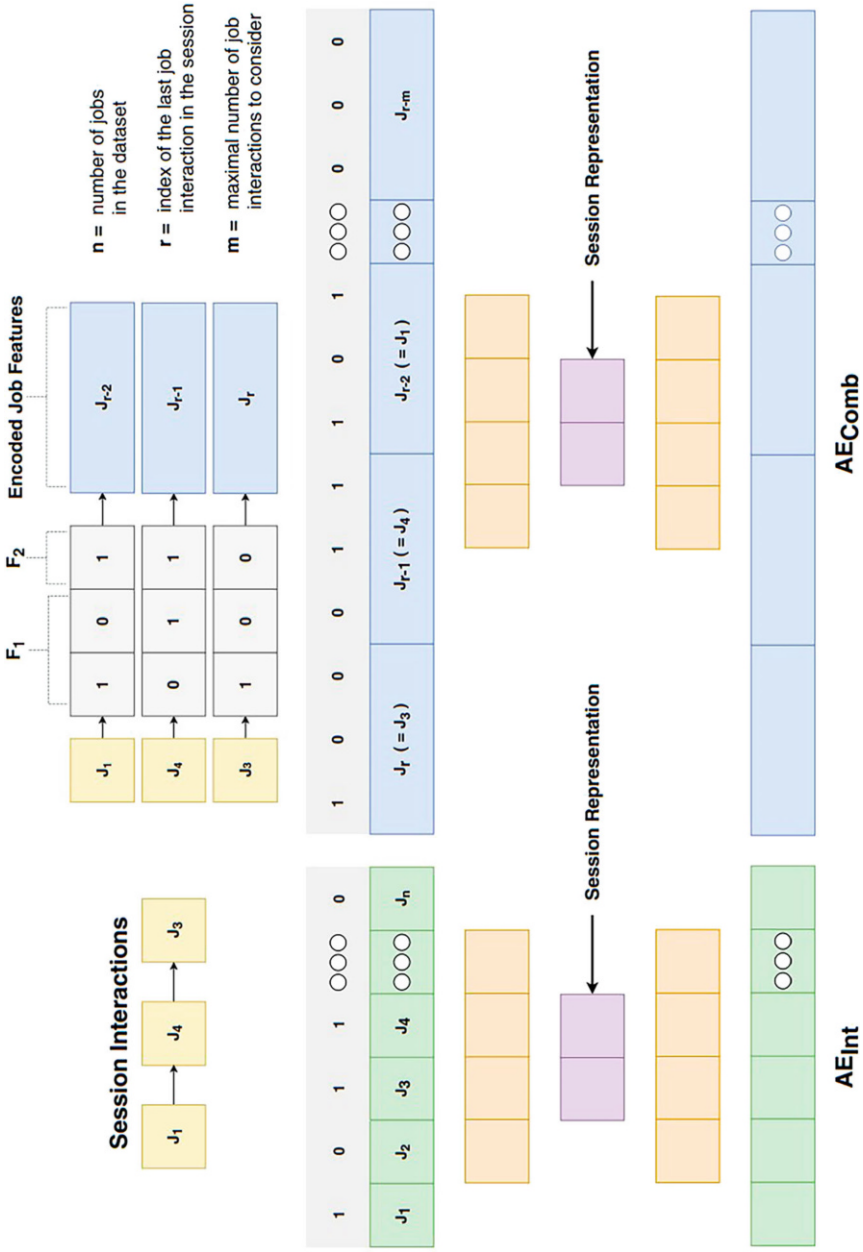

Fig. 4.12 Modeling job sessions in [9]

uses the most recent m job interactions within the session and generates a binary encoding of the job content features in descending order.

Deng et al. proposed a graph-based approach, HybridGNN-SR, and improved the model's efficiency from two aspects [16]. On the one hand, it reconstructs the mutual information of each session graph to predict recommendations, and on the other hand, it presents a session based on higher conceptual representation by considering the adjacent items in a session. In HybridGNN-SR, graph learning is performed based on the combination of two supervised and unsupervised methods to provide item transition patterns in a session from the perspective of the graph. In the unsupervised learning part of the graph, variational graph autoencoders are combined with mutual information to represent the graph nodes of a session. In the supervised learning part, a routing algorithm is used to extract high-level conceptual features from the session, which considers the dependencies between items in the session. This model extracts correlation information of dependent items and then feeds them to the routing mechanism to extract the diversity of user interests from a session.

Another session-based recommender system has been proposed by Kang et al. that combines a bidirectional encoder and an autoregressive decoder and uses noisy transformation for user interactions [20]. E-BART4Rec (Entangled BART for Recommendation) is built upon BART, a model that is widely used in NLP tasks. BART uses a left-to-right decoder and injects noise into its bidirectional encoder, which can reduce the gap between training and inference. In E-BART4Rec, the encoder network consists of two sub-layers, a multi-head attention-based network and a pointwise feedforward network. The decoder consists of three sub-layers, masked multi-head attention-based network, multi-head attention-based network, and pointwise feedforward network. The gating mechanism is used in E-BART4Rec to detect the importance of the decoder or encoder to calculate the next interaction. Unlike other similar methods, the output of E-BART4Rec dynamically integrates a bidirectional encoder and an autoregressive decoder based on the gating mechanism and according to the characteristics of user interactions.

4.4 Session-Based Recommender System Using GAN

Before looking at the approaches of generative adversarial network models in session-based recommender systems, an overview of GAN and the reasons that made it an effective choice for SBRS are provided.

4.4.1 Why GAN?

The significant successes achieved using generative adversarial networks (GANs) in various fields of deep learning have been the reason for their use in recommender

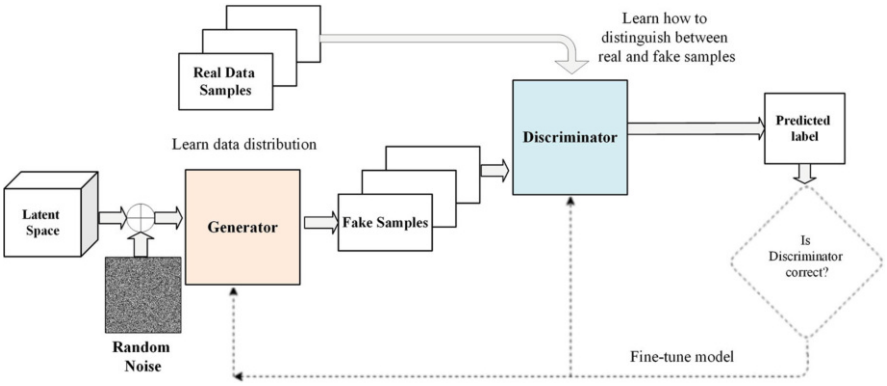

Fig. 4.13 The general architecture of GAN

systems. GAN-based recommender systems will become very popular in the field [46]. This popularity is because the GAN concept provides new opportunities to reduce data sparsity and noise. Several existing studies have confirmed the effectiveness of GAN-based approaches and minimax game theory in the objective function to reduce data noise. Other studies have also tried using the discriminator to distinguish informative examples in an adversarial manner. Meanwhile, to address the issue of data sparsity, a separate line of research has explored the capabilities of GANs to generate user profiles with augmenting user-item interaction and auxiliary information. Due to the significant impact of GAN networks on recommender systems, these networks have recently been utilized in session-based recommender system, some of which are discussed in this section.

Generative adversarial networks are a computational method based on game theory, and there is a combination of two neural networks at their core, the generator and the discriminator. There is also an adversarial relationship between these two networks. In simple words, the first one is trained to generate data, and the second is trained to distinguish between real and fake data. The generator, who is a forger, tries fooling the discriminator network, which is a detective. The detective's goal is to distinguish the fake sample from the real one, and with each unsuccessful attempt, it optimizes its operations by receiving feedback from it. Figure 4.13 shows the general architecture of the GAN.

The goal of the generative adversarial network is to learn a generative model G that can generate samples from the data distribution p_x by transforming the latent vector Z into samples in the data space with higher dimensions x. Usually, latent vectors are sampled from Z using a uniform or normal distribution. To train the generator model G, a discriminator model D is trained to distinguish the real training samples from the fake ones generated by G. Therefore, the discriminator model returns a value $D_x \in [0,1]$, which can be interpreted as how likely the input sample x is a true sample from the data distribution. In this configuration, the generative model is trained by generating examples most similar to the real training examples.

However, the discriminator model is continuously trained to distinguish real samples from fake ones.

The loss function in the generative adversarial network is defined as minimax. Thus, if the goal of training the discriminator model is to bring the loss function $V(D, G)$ to the minimum value, the goal of training the generator model is to bring the same loss function to the maximum value (or to minimize the discriminator's reward). According to the structure of the discriminator model, the loss function of the generative adversarial network can be determined by Eq. (4.24):

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{4.24}$$

In the above equation, G and D represent the generator and the discriminator models, respectively, $p_{\text{data}}(x)$ is the probability distribution of the real data, $p_z(z)$ is the probability distribution of generator data, E_x is the expected value over all real data instances, and E_z is the expected value over all random inputs to the generator (the expected value over all generated fake instances $G(z)$).

According to the loss function, a generative adversarial network is trained in two stages. First, the loss function for the discriminator network is calculated based on the training data x and the output of the generator network z according to Eq. (4.25). Then, the chain derivatives for the training parameters of the discriminator model are calculated based on these values and updated according to Eq. (4.26) by the gradient ascent method:

$$E_{D(x,G(z))} = E_x[\log D(x)] + E_z[\log(1 - D(G(z)))] \tag{4.25}$$

$$\theta_{D(k+1)} = \theta_{D(k)} + \eta \frac{\partial E_D}{\partial \theta_D}(k) \tag{4.26}$$

After updating the values related to the discriminator model, the value of the loss function of the generator model is calculated according to Eq. (4.27), and then its weights are updated using the gradient descent method according to Eq. (4.28).

$$E_{G(z)} = E_z[\log(1 - D(G(z)))] \tag{4.27}$$

$$\theta_{G(k+1)} = \theta_{G(k)} - \eta \frac{\partial E_G}{\partial \theta_G}(k) \tag{4.28}$$

In summary, GANs are proposed to avoid many disadvantages associated with other generative models [50]:

- They can generate samples in parallel instead of using a proportional runtime to the dimensions of x.
- The design of the generator's performance has very few limitations. This is an advantage over Boltzmann machines, for which few probability distributions accept tractable Markov chain sampling.

- Markov chain is not required. This is an advantage over Boltzmann machines and random generator networks.
- No variational bounds are required, and the types of models that can be used in the GAN framework are already known to be universal approximators, so GANs are asymptotically consistent.
- GANs inherently produce better samples than other models.

Despite the proposed advantages of GAN, there are also limitations to this model. Although several studies discuss the convergence and existence of Nash equilibrium in the GAN game, GAN training is very unstable and hard to converge. GAN solves a minimax game iteratively through the gradient descent method for the generator and discriminator. From the loss function approach, a solution to the GAN game is a Nash equilibrium, which is a point of parameters where the cost of the discriminator and the generator are minimal with respect to their parameters. However, decreasing the loss function of the discriminator can increase the loss function of the generator and vice versa. Therefore, the convergence of the GAN game may often fail and is prone to instability.

Another important issue for GAN is the mode collapse problem. This problem is harmful to GANs applied in real applications because mode collapse limits the GAN's ability to diversity. The generator must trick the discriminator, not describing the multimodality of real data distribution. Mode collapse also can occur even in a simple experiment, which prevents the usage of GANs due to low diversity. However, for a highly complex and multimodal real data distribution, mode collapse is still a problem that GANs should solve.

Generative adversarial networks have been used in any research related to text and image generation, feature extraction, etc. GAN could map input variables to another feature space by using min-max optimization and discriminative models. It can be used to solve problems of sparse datasets and create effective recommender systems with acceptable performance. Recently, generative adversarial networks have been widely used in the field of deep learning, and based on their capabilities, different deep generative approaches have been presented in the field of session-based recommender systems.

The purpose of generative adversarial networks in recommender systems is to reduce casual and malicious noises in data and to increase the ability to recognize samples from unobserved items. Additionally, generative adversarial networks have been considered and used in all kinds of recommender systems due to their ability to reduce the problems caused by the spareness of datasets. In the following subsection, we discuss several research performed using GAN in SBRS.

4.4.2 GAN Approaches

One of the first methods that used generative adversarial networks in session-based recommender systems has been proposed by Zhao et al., called Prioritizing

Long- And Short-Term Information in top-n reCommendation (PLASTIC) [32]. PLASTIC used matrix factorization approaches and recurrent neural networks along with generative adversarial networks to recommend the top N items to the user. The proposed model adaptively adjusts how to combine short-term and long-term information about users and items. In the process of adversarial training, the generator model takes users and items as input for predicting the list of user recommendations. For the discriminator model, it integrates prioritizing short- and long-term models through the Siamese network as real samples are correctly distinguished from generated samples.

Bharadhwaj et al. proposed another SBRS based on a generative adversarial network, called recurrent generative adversarial network (RecGAN) [30]. RecGAN, unlike other similar methods, is based on time, and it learns the latent temporal feature of the user and item under the framework of generative adversarial networks to improve the efficiency of the recommender system. RecGAN combines generative adversarial networks with recurrent neural networks to learn temporal features. The generator model specifies the distribution of items to predict a sequence of items for a user. In fact, the generator model in RecGAN is designed to learn the distribution of the relevance of items for users and extract an unlabeled sequence from the relevant items generated to obtain a better estimate of their relevance. The discriminator model also determines whether the sampled items correspond to the users' true interest distribution. RecGAN modifies the GRU cell so that it can obtain latent factors of users and items that are observable from short- and long-term profiles.

Session-based recommender systems that use reinforcement learning have two limitations: (1) Ignore the user's skip behaviors that are scattered in sequential patterns. (2) When the positive feedbacks in the dataset are sparse, the system cannot use positive and negative feedback together. To solve these two problems, Gao et al. utilized reinforcement learning along with generative adversarial networks in a session-based interactive recommender system, called DRCGR [26]. In DRCGR, convolutional neural networks and generative adversarial networks use deep Q-Network learning to better understand high-dimensional data. CNN is used to detect the sequential features of positive user feedback, and GAN is used to learn negative feedback representations. Finally, the negative and positive feedbacks are simultaneously sent to the deep Q-Network to create a better action-value function. This increases the robustness of session-based interactive recommender systems.

The sequential interests of the user are first detected in DRCGR using a deep model based on convolutional neural networks. This is performed using convolutional filters. The spatiotemporal sequences of user clicking behavior and their corresponding items are embedded in a potential dimensional space, and then the horizontal convolution layer and vertical convolution layer are used to learn the local features of the above images. Then, in generative adversarial networks, the generator model purposefully generates different personalized negative samples, and the positive feedback for training comes from the random sampling of real click data. The discriminator model also recognizes real samples from irrelevant and non-real samples.

Another session-based recommender system has been proposed by Chen et al. using the modeling of the complex behavior of users, which has utilized reinforcement learning and generative adversarial networks [27]. In this system, user dynamic behavior modeling and related reward functions are learned in a unified minimax framework, and then reinforcement learning policies are learned to use the model. Using the user model as the simulation environment, a cascading Q-Network is presented for a combinatorial recommendation policy, which can control many candidate items well and reduce computational complexity. One of the advantages of using the generative adversarial network in this method is that it improves the representation of the user model as well as the reward function according to the user's learned model and considers online adaptation for new users.

In session-based recommender systems, the user's previous behaviors are used to predict his next behaviors, which could be performed using reinforcement learning methods. However, reinforcement learning methods may experience unstable convergence in learning processes. In fact, these methods require many training data, which is challenging in sparse session-based recommender systems. For this reason, the use of the generative adversarial network model is very effective. On the other hand, the generation of negative samples in this type of system should not be random because an item that the user likes but has no interaction with it may be considered a negative sample for that user randomly. Therefore, the policy used in the generator model is critical in generative adversarial networks.

For better processing of users' immediate feedback in session-based recommender system based on the collaborative filtering approach, Zhao et al. combined reinforcement learning and a generative adversarial network, called Deep Generative Adversarial Networks-based Collaborative Filtering (DCFGAN) [25]. DCFGAN exploits the immediate feedback of users, which solves the need for information and training examples. On the other hand, the generated negative samples are optimized using collaborative filtering to provide sufficient recommendations to users. To solve the instability of the training process caused by the uncertain probability in the policy gradient algorithm in the previous session-based recommender system, the deep deterministic policy gradient (DDPG) algorithm is proposed to increase the stability of the training process. Meanwhile, DDPG optimizes the value function and reduces the number of iterations required for convergence. This method uses pre-training collaborative filtering to negative sample items with low user interest, effectively improving negative sampling accuracy, which is more suitable for recommendation scenarios. The generator model of GAN is also improved using repeated experiences. Figure 4.14 shows the architecture of the DCFGAN.

Considering the high potential of generative adversarial networks in improving the efficiency of recommender systems based on collaborative filtering, Ren et al. developed a multi-factor generative adversarial network (MFGAN) [33]. MFGAN uses adversarial training to decouple factor utilization from the sequence prediction component. This provides more flexibility in the use of external contextual information in sequential recommendations, which can improve the interpretability of recommendations. Two main modules are used in MFGAN: (1) transformer-based

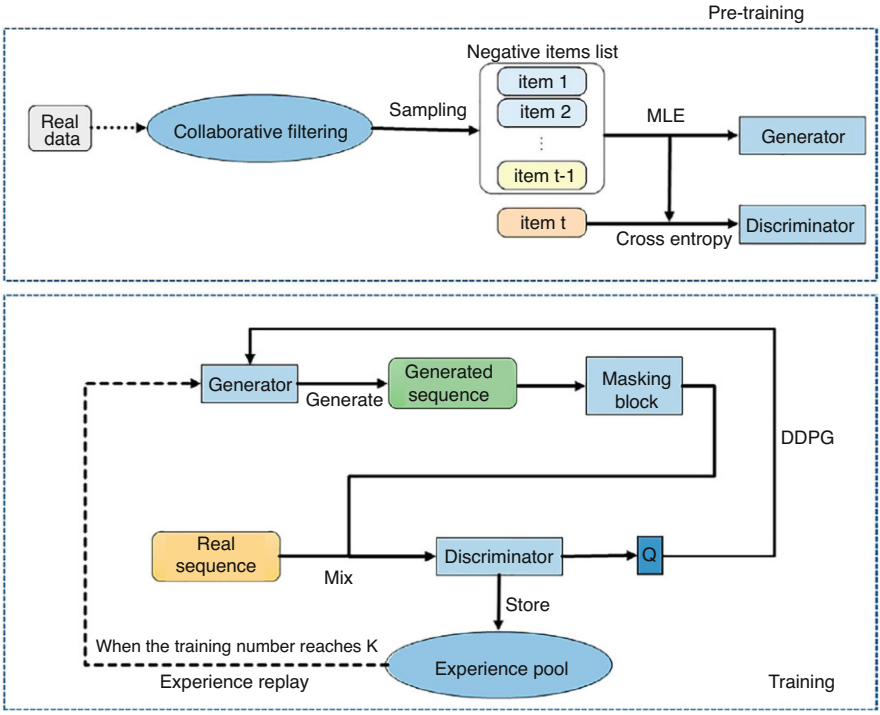

Fig. 4.14 The architecture of the DCFGAN [25]

generator module and (2) multiple factor-based discriminator module. Based on data related to user-item interactions, the generator model predicts the next proposed items, and the discriminator model considers the sequence of recommendations based on various factors of the available information. The discriminator uses a bidirectional transformer and can refer to the information on subsequent positions for evaluations and increase the reliability of the evaluation. Because of the discrete nature of the item generation process, the training of MFGAN is performed through reinforcement learning. In this method, different factors are separated from the generator and used by the discriminator to extract reward signals to improve the generator. The architecture of MFGAN is shown in Fig. 4.15.

4.5 Session-Based Recommender System Using FBM

Before looking at the approaches of flow-based models in session-based recommender system, an overview of FBM and the reasons that made it a suitable choice for SBRS are provided.

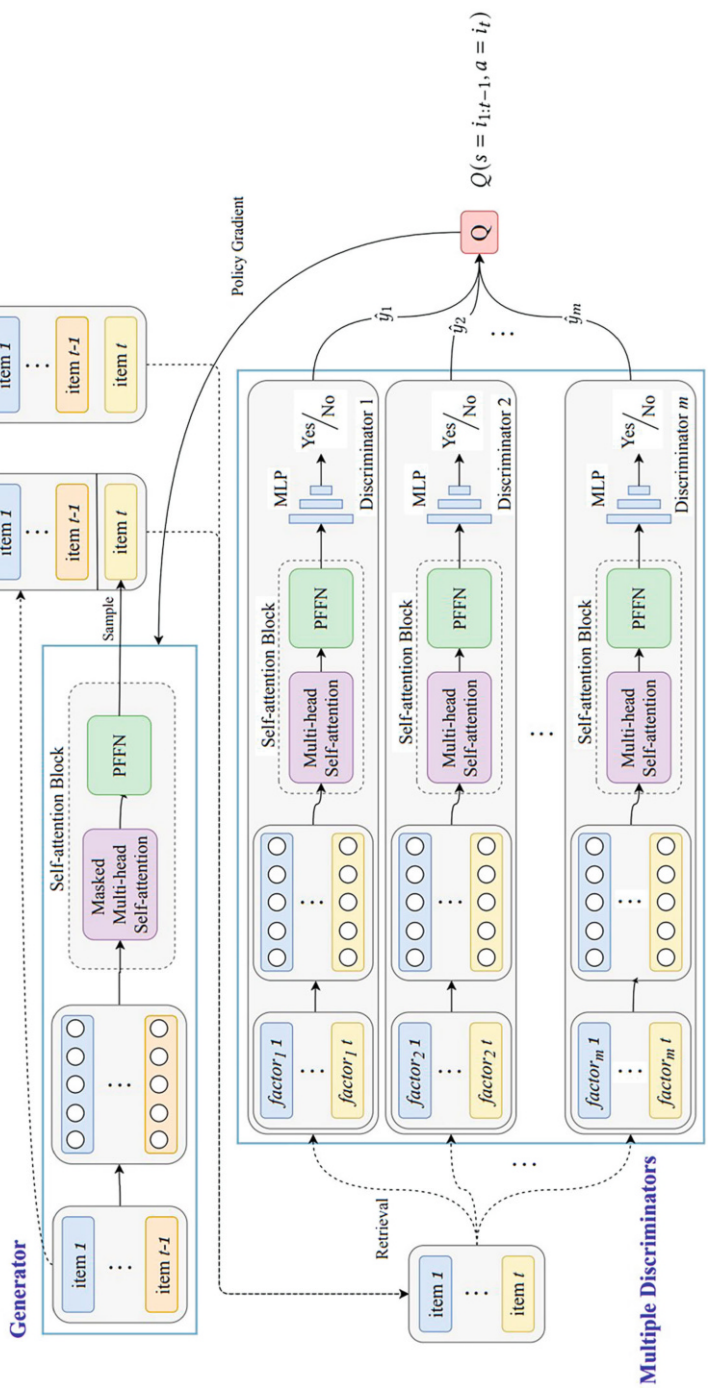

Fig. 4.15 The architecture of the MFGAN [33]

4.5.1 Why Flow-Based Models?

Despite using Bayesian inference and uncertainty representation, collaborative VAE models are difficult to optimize, mainly due to inherent biased variational inference. Models based on autoencoders usually assume that the posterior can be decomposed into several independent factors, while intractable variational inference requires searching for the best approximation of the true posterior in a parameter family of distributions that are usually specified in advance.

However, distributional models are inflexible enough to match the true posterior and recommendation uncertainty unless a precise family of distributions is chosen. Such challenges motivate researchers to enrich the variational posterior distribution by using normalizing flow [38], which is a set of invertible transformations to the desired variables with a simple initial distribution. Compared to VAE and GAN, flow-based models have so far attracted less attention, specially in session-based recommender systems, although their unique advantages, such as accurate latent variable inference and analytical likelihood evaluation, can propose great potential for future work.

Therefore, flow-based models are used to approximate the real posterior of stochastic latent factors, which can significantly reduce the inference bias in the VAE-based models and, as a result, improve the accuracy of predicting the next click. In fact, using richer posterior distributions can effectively reduce the gap between the approximate posterior and the true posterior. This approximate gap is caused by the encoding cost, which is mainly due to the incorrect assumption of the probabilistic distribution.

Flow-based models are built with a sequence of invertible transformations, and unlike the previous two models, they explicitly learn the data distribution p(x), and therefore, the loss function is simply a negative log-likelihood. Figure 4.16 shows the main idea of flow-based models. In each step, using the change of variable theorem, a new variable is replaced. Finally, the final distribution obtained can be close enough to the target distribution.

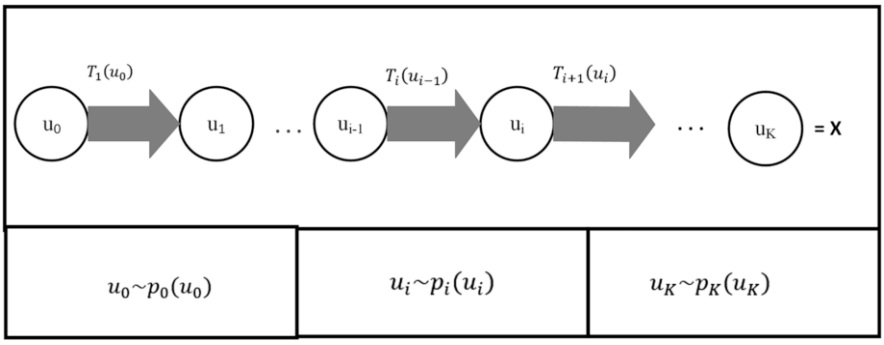

Fig. 4.16 Transforming a simple distribution into a complex distribution by applying a sequence of invertible transformations

Normalizing flow-based models provide a general approach for constructing flexible probability distributions over continuous random variables. If x is a d-dimensional real vector, and suppose we want to define a joint distribution on x, the basic idea of flow-based modeling is to express x as a transformation function T of a real vector u sampled from $p_u(u)$:

$$x = T\ (u) \quad \text{where} \quad u \sim p_u(u). \tag{4.29}$$

$p_u(u)$ is the base distribution of the flow-based model [7]. The transformation function T and the basic distribution $p_u(u)$ can have their own parameters. The specific characteristic of flow-based models is that the transformation function T must have an inverse function and T and T^{-1} must be differentiable. In this situation, the density x is well defined and can be obtained using the change of variable theorem by Eq. (4.30):

$$p_x(x) = p_u(u)|\det J_T(u)|^{-1} \qquad \text{where} \qquad u = T^{-1}(\mathrm{x}) \tag{4.30}$$

Equivalently, we can also write $p_x(x)$ in terms of the Jacobian T^{-1} by Eq. (4.31):

$$p_x(x) = p_u\left(T^{-1}(x)\right)\left|\det J_{T^{-1}(x)}\right| \tag{4.31}$$

The Jacobian $J_T(u)$ is the $D \times D$ matrix of all partial derivatives of T given by Eq. (4.32):

$$J_T(u) = \begin{bmatrix} \frac{\partial T_1}{\partial u_1} & \cdots & \frac{\partial T_1}{\partial u_D} \\ \frac{\partial \dot{T}_D}{\partial u_1} & \cdots & \frac{\partial \dot{T}_D}{\partial u_D} \end{bmatrix} \Bigg(\tag{4.32}$$

In practice, a flow-based model is often constructed by implementing T (or T^{-1}) with a neural network and taking $p_u(u)$ as a simple density (such as a multivariate normal).

4.5.2 Flow-Based Approaches

Many recommender systems based on generative models, such as variational autoencoders, are very effective for learning non-linear user-item representations in collaborative filtering-based approaches [17, 47, 48]. However, session-based recommender systems cannot use these models directly for the following reasons:

- Data availability: Lack of access to user profile information and long-term interactions of users.

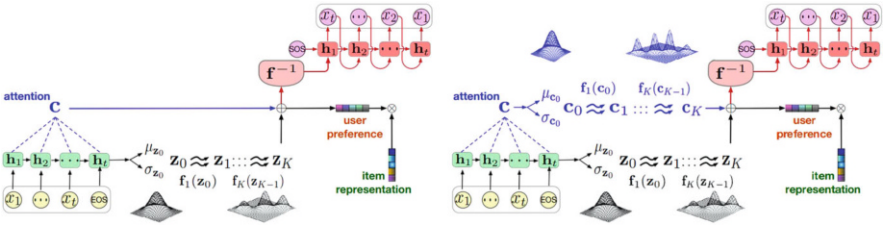

Fig. 4.17 The architecture of VASER [2]

- Bypassing challenge: Autoregressive models combined with soft attention-based mechanisms are capable of reconstructing encoded sessions. This mechanism may weaken the effects of latent factors and potentially reduce the performance of the model using variational autoencoders.
- Biased inference: Models based on an autoencoder assume a predefined prior probability for latent factors that are restrictive for learning the distribution of data and may lead to the approximate deviation of the inferred posterior from the true distribution.

To this end, several research have been proposed to utilize flow-based models to overcome the mentioned problems. Zhong et al. proposed an SBRS framework using Bayesian inference for flexible parameter estimation, called VAriational SEssion-based Recommendation (VASER) [2]. In VASER, instead of directly applying extended variational autoencoders, the normalizing flow method is introduced to estimate the probabilistic posterior. While maintaining Bayesian inference of variational autoencoders, the VASER model also allows for the exploration of non-linear probabilistic latent variable models. This method augments session-based recommender systems that have used recurrent neural networks with stochastic latent variables trained by stochastic and amortized variational inference, making it possible to infer a stable and effective approximation of a high-level objective of the whole session from the observed clicks. To encode more useful information in latent variables, an auxiliary factor is introduced that leverages variational attention on user clicks. Unlike deterministic attention, the proposed attention mechanism can accurately model click sessions without overpowering the latent representation. In addition, the normalizing flows are used to approximate the real posterior of stochastic latent factors, which can significantly reduce the inference biases in the proposed models based on a variational autoencoder and improve the accuracy of next-click prediction.

Zhong et al. also developed two variants of VASER with a deterministic attention mechanism (VASER-DA) and with a variational attention mechanism (VASER-VA), as shown in Fig. 4.17. Each model includes two main components, which are the GRU and the attention modules. The GRU component takes sequential interests whose hidden state can extract non-linear interests. The attention component is used to enhance the GRU network, which dynamically selects different parts of the input and combines them linearly. VASER-DA uses a deterministic attention mechanism,

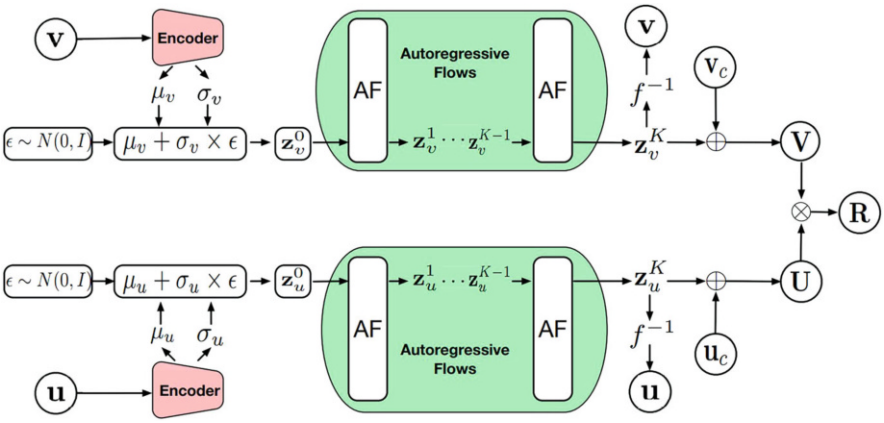

Fig. 4.18 The architecture of the CAF [35]

while VASER-VA uses the attention vector as a stochastic latent factor to overcome the bypassing phenomenon caused by recurrent neural networks and the deterministic attention mechanism. For flexible posterior approximation, both models utilize normalizing flows.

Considering the improvement in the VASER for using variational autoencoders and flow-based models in session-based recommender systems with a collaborative filtering approach, the variation distribution choices are still not enough to recover and improve the true distributions, and this model is difficult to optimize. The reason for this problem is the biased maximum likelihood estimates of the model parameters. By choosing only one family of probability distributions, the models are inflexible enough to match the true posterior and the uncertainty of recommendations. To this end, Zhou et al. extended the flow-based model to CF for modeling implicit feedbacks and presented the collaborative autoregressive flows (CAF) [35]. The CAF transforms a simple initial density into more complex densities through a sequence of invertible transformations until a desired level of complexity is reached. This non-linear probabilistic approach provides the possibility of representing the uncertainty and accurate tractability of latent variable inference in item recommendations. In CAF, using the flow function to approximate the true posterior probability of the latent factors, it adopts the probabilistic density estimation to reduce the inference bias in the existing Bayesian recommendation model to improve the recommendations. The combination of two autoregressive flows gives the CAF the efficiency of variational inference and sampling and fills the gap between latent factors with simple base distribution and real data with a complex distribution. Figure 4.18 shows the schematic of CAF.

4.6 Discussion

Recently, deep generative models have been effective in the efficiency of session-based recommender systems. The deep generative models help reduce problems caused by complex dependencies between variables in different time steps or different sessions. On the other hand, generative methods have the ability to generate more samples for model training and can help reduce the problems caused by data sparsity. The proposed approaches discussed in this chapter provide models of session-based recommender systems that have utilized deep generative techniques.

Autoencoders (AE), generative adversarial networks (GAN), and flow-based models (FBM) are among the most widely used deep generative models that have been used in session-based recommender systems. A large percentage of proposed approaches using AE belong to variational autoencoders (VAE), an essential type of AEs. The difference between variational autoencoders and generative adversarial networks is their learning process and training time. Variational autoencoders are a semi-supervised method, while generative adversarial networks are an unsupervised method. Conversely, the training time of generative adversarial networks is more than variational autoencoders.

Approaches based on autoencoders usually create a latent presentation of input data, which in most cases includes user interactions and transactions in sessions. For example, one of these approaches is the NARM method, which models the sequential behavior of users based on the encoder and decoder architecture, determines the user's main goal in the current session, and finally creates an integrated representation of the session [18]. NARM considers only one session, which may have noise or a set of choices of several users; therefore, all dependencies may not be recognized correctly. Other methods, such as [15, 19, 22], have been presented to improve NARM. For example, in [15], the size of the inputs, the bottleneck of the latent presentation, and the outputs are changed, and in [19], parallel neural networks are used to provide recommendations, so in addition to the information of the current session, the information of the neighboring sessions is also employed. The proposed method in [22] also uses a dual attention-based neural network and GRU-based hybrid encoder to reduce the problem of ignoring the user's interest in previous sessions.

As discussed before, variational autoencoders are widely used in session-based recommender systems. For example, in [9], a session-based job recommender system investigated three types of autoencoders, including classic autoencoder, denoising autoencoder, and variational autoencoder, to provide sessions, and the best result was related to the variational autoencoder. Many recommender systems based on deep generative models, such as variational autoencoders, have provided very effective results for learning non-linear user-item representations in collaborative filtering methods. For example, in [2], variational autoencoders have been extended for modeling users' implicit feedback in session-based recommender system and through augmenting RNN by stochastic latent variables trained by stochastic and amortized variational inference allow effective inference of the entire

session from the observed clicks. Meanwhile, it uses normalizing flows to approximate the real posterior of stochastic latent factors, greatly reducing the inference biases in the proposed variational autoencoder-based models and improving the accuracy of next-click prediction. Still, in [2], the variation distribution choices are insufficient to recover and improve the correct distributions, making this model difficult to optimize. The main reason for this problem is the inherent biased variational inference. To reduce this problem, the method proposed in [35] using autoregressive flows approximates the true posterior probability of stochastic latent factors; enables flexible and tractable probabilistic density estimation, greatly reducing the biased inference in existing Bayesian proposed models; and improves the accuracy of the recommender systems. To this end, the proposed method [17] is based on a sequence of variational autoencoders that use variational autoencoders to model user interests along with latent variables and time dependencies.

An example of a denoising autoencoder in the session-based recommender system is the proposed system [14], which uses denoising autoencoders to provide a unique representation of news articles, along with the categories of articles that increase the system's efficiency. Additionally, in [20], a new recommender system is proposed that combines a bidirectional encoder and an autoregressive decoder and uses a noisy transformation for user interactions.

The purpose of generative adversarial networks (GAN) in session-based recommender system is to reduce casual and malicious noises in data and increase the ability to recognize samples from unobserved items. In addition, GANs have been considered in all kinds of recommender systems due to their ability to reduce the problems caused by the sparsity of datasets. The proposed method [32] combines matrix factorization approaches and RNNs with GANs and integrates prioritizing long-term and short-term models through Siamese networks. Another session-based recommender system based on time-based GANs has been proposed in [30], which combines GANs with RNNs to learn the latent temporal features of the user and item.

Considering the high potential of generative adversarial networks in improving the efficiency of recommender systems based on collaborative filtering, the method presented in [33] uses generative adversarial networks with multiple factors and uses adversarial training to decouple factor utilization from the sequence prediction component. This provides more flexibility in the use of external contextual information in sequential recommendations, which can improve the interpretability of recommendations. Several research presented in the field of session-based recommender system with collaborative filtering approach have also used reinforcement learning in addition to generative adversarial networks [25–27]. In [26], convolutional neural networks and generative adversarial networks use deep Q-Network learning to better understand high-dimensional data. In [27], reinforcement learning and generative adversarial networks are combined, which handle a large number of candidate items well and reduce the computational complexity. The advantage of using a generative adversarial network in this method is to improve the representation of a user model and reward function according to the user's learned model and considers an online adaptation for new users. With the aim of processing

users' immediate feedback as best as possible, a method is proposed in [25] that combines Q-Learning with actor-critic models in reinforcement learning. In this method, the combination of generative adversarial networks and reinforcement learning is used to exploit the immediate feedback of users, which solves the need for information and training examples. On the other hand, the generated negative samples are optimized using collaborative filtering to provide better recommendations to users.

At the end of this part, an important point about using flow-based models in session-based recommender system should be mentioned. Normalizing flow models are successful at estimating high-dimensional densities; however, their process still has some disadvantages. Because the latent space where input data is projected is not lower-dimensional, flow-based models do not allow data compression by default. These models cannot estimate the probability of the samples not being from the same distribution as the training set. An important feature of normalizing flows is the invertibility of their bijective map. This property guarantees the theoretical inversion and is achieved with some restrictions on the design of the models. Inverse integration is important in order to ensure the applicability of the change of variable theorem, the Jacobian computation of the map, as well as the sampling with the model. However, this invertibility is violated in practice, and the inverse map explodes due to numerical imprecision.

For future research based on deep generative models discussed in this chapter, the following items can be considered:

- A critical problem in recommender systems is data sparsity, which means that the values of the user-item matrix are limited. This problem can be solved using side information. Choosing appropriate auxiliary information to help understand the relationship between users and items is essential to further improve recommendation accuracy. Furthermore, there is little work examining changes in users' interests or intentions. The autoencoder's ability to process data from heterogeneous sources brings more opportunities to recommend various items based on unstructured data, such as textual, visual, audio, and video features. In fact, cross-domain models help represent the target domain using knowledge learned from other sources and can be a suitable option to deal with the problem of data sparsity. A topic that has been widely studied in cross-domain recommendations is transfer learning [25]. This study improves learning ability in the target domain by using knowledge transferred from other domains. However, integrating information from different domains into the same representation space is still a challenging problem. The GAN can continuously learn and optimize the mapping process from the source domain to the target domain, thereby enriching the training data of the recommendation model.
- Fusion methods can model the heterogeneous characteristics of determinant factors, such as user, items, and context, in recommender systems. Studies have integrated deep generative models with traditional recommendation methods. However, only a few limited studies have been conducted in this field that combined the use of deep generative models such as autoencoder and generative

adversarial networks with other deep learning methods. For example, an autoencoder can be combined with deep semantic similarity models to learn the semantic representation of items in a common continuous semantic space and to measure their semantic similarities. Therefore, the fusion models are a promising area for future research that is largely unexplored and where more studies are expected.

- Attention-based mechanisms are an intuitive but effective technique that can be applied to deep neural networks. The attention mechanism provides a good solution to deal with long-term dependencies and helps the network remember inputs better. By applying the attention mechanism, session-based recommender system using deep generative models can filter meaningless content and select the most meaningful ones that provide good interpretability.
- The time complexity of session-based recommender system based on deep generative models is relatively high. For example, GANs include two generator and discriminator components, and each part can be composed of several neural network layers. This problem becomes more serious when the model contains several generators and discriminators. The appropriate use of generative and discriminative neural networks and the efficient training process in a mutually promoting mode, at the same time considering suitable feedback between two generative and discriminative modules, can be an effective solution to improve the performance of the final model.
- Scalability is critical for session-based recommender system models because the ever-increasing volume of data and the introduction of the concept of big data in this field make time complexity one of the main considerations. Deep generative models such as GAN and autoencoders have been applied to some commercial products. However, due to the continuous improvement of GPU computing power, further research on recommender systems based on deep generative models is needed in the following areas: (1) incremental learning of streaming data that is generated at high speed (velocity factor in big data concepts), such as cases where numerous interactions between users and items are performed; (2) accurate calculation of tensors with high dimensions, specially when using multimedia data sources; and (3) balancing the complexity and scalability of the model when we have exponential parameters' growth in the developed model.

In many cases, the research fields mentioned in the above paragraphs can be considered together or have mutual effects on each other. For example, how to develop a session-based recommender system based on deep generative models that can simultaneously consider high scalability and the effective combination of auxiliary information can be one of the valuable directions for future research.

Table 4.5 summarizes the existing works discussed in this chapter and addresses the application domain, deep learning model, type of input data, embedding technique, and loss function of each approach.

Table 4.5 A summary of the reviewed research

Ref.	Domain	Deep learning model	Input data	Embedding technique	Loss function
[18]	E-commerce	Encoder/ decoder	Session click	One-hot encoding	Cross-entropy
[19]	E-commerce	Autoencoder	Items of session	One-hot encoding	Cross-entropy
[22]	E-commerce	Autoencoder	Session click	One-hot encoding + encoder	Bi-linear similarity function
[15]	Trip	Autoencoder	User features, cities, event time, categories	Autoencoder + space-time encoding	Focal loss function
[14]	News	RNN (LSTM/ GRU)	News articles	Denoising autoencoder	Proposed loss function
[17]	Movie	Variational autoencoders	User preferences	Binary matrix	Proposed loss function
[21]	Movie	Autoencoder	Sessions	One-hot encoding	Cross-entropy
[9]	Job	Autoencoder (denoising/ variational)	Sessions	Binary-encoded representation + autoencoder	Kullback-Leibler divergence
[16]	E-commerce	Variational graph autoencoder	Items of session	D-dimensional latent feature vector	Cross-entropy + proposed unsupervised loss function
[20]	Movie, music	Autoencoder	User interactions and items	D-dimensional vector	Negative log-likelihood
[32]	Movie	GAN + matrix factorization + RNN	Users and items of session	D-dimensional vector	Minimax loss
[30]	Fitness, movie	GAN + RNN	User ratings	Temporal preference vector representation	Minimax function
[26]	E-commerce	GAN + DQN	User's recently clicked item	Embedding lookup + convolutional operation	Proposed loss function
[27]	E-commerce, news, movie	GAN + cascading Q-Networks	User's historical interactions	D-dimensional vector	Proposed loss function
[25]	E-commerce	GAN + reinforcement learning	User's historical interactions	Real-valued vector	Proposed loss function
[33]	Movie, music, game	GAN	Items of session	Item embedding matrix	Proposed loss function
[2]	E-commerce	Normalizing flow + GRU	Session	D-dimensional latent representation	ELBO + cross-entropy
[35]	Movie, music, article	Autoregressive flow	Matrix of users and items	Binary matrix	Proposed loss function

4.7 Conclusion

One of the most widely used methods of deep learning in session-based recommender system is deep generative models. The deep generative models have the ability to generate more samples for model training and reduce the problems caused by data sparsity. The recommender systems based on deep generative models help reduce problems caused by complex dependencies between variables in different time steps or different sessions. The proposed approaches discussed in this chapter provide models of SBRS that utilize deep generative techniques. Most of the research presented in this field is based on autoencoder, generative adversarial network, and flow-based models.

In SBRS, various types of autoencoders, such as variational autoencoders, convolutional autoencoders, denoising autoencoders, etc., have been used, but a large percentage of proposed approaches have employed variational autoencoders that are very effective for learning non-linear user-item representations. Generative adversarial networks have been considered in session-based recommender systems due to their ability to reduce the problems caused by data sparsity. Indeed, it alleviates casual and malicious noise in data to increase the ability to recognize samples from unobserved items. Compared to autoencoders and generative adversarial networks, flow-based models have so far attracted less attention in the field of session-based recommender systems, although their unique advantages, such as accurate latent variable inference and analytical likelihood evaluation, can provide great potential for future work.

This chapter concluded with several discussions on the proposed research and provided future directions and trends in session-based recommender systems using deep generative learning models.

References

1. Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet A. Orgun, and Defu Lian. "A survey on session-based recommender systems." ACM Computing Surveys (CSUR) 54, no. 7 (2021): 1-38. https://doi.org/10.1145/3465401
2. Ting Zhong, Zijing Wen, Fan Zhou, Goce Trajcevski, and Kunpeng Zhang. "Session-based recommendation via flow-based deep generative networks and Bayesian inference." Neurocomputing 391 (2020): 129-141. https://doi.org/10.1016/j.neucom.2020.01.096
3. Zhidan Wang, Wenwen Ye, Xu Chen, Wenqiang Zhang, Zhenlei Wang, Lixin Zou, and Weidong Liu. "Generative session-based recommendation." In Proceedings of the ACM Web Conference 2022, Lyon France April 25 - 29, 2022, pp. 2227-2235. https://doi.org/10.1145/3485447.3512095
4. Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models." IEEE transactions on pattern analysis and machine intelligence (2021). https://doi.org/10.1109/TPAMI.2021.3116668

5. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. "A Neural Probabilistic Language Model." Journal of Machine Learning Research 3 (2003): 1137-1155. https://dl.acm.org/doi/10.5555/944919.944966
6. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio "Generative adversarial networks", Communications of the ACM, Volume 63, Issue 11, 2020, pp 139–144, https://doi.org/10.1145/3422622
7. George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. "Normalizing flows for probabilistic modeling and inference." The Journal of Machine Learning Research 22, no. 1 (2021): 2617-2680. https://dl.acm.org/doi/10.5555/3546258.3546315
8. Yang Song, and Diederik P. Kingma. "How to train your energy-based models." arXiv preprint arXiv:2101.03288 (2021). https://doi.org/10.48550/arXiv.2101.03288
9. Emanuel Lacic, Markus Reiter-Haas, Dominik Kowald, Manoj Reddy Dareddy, Junghoo Cho, and Elisabeth Lex. "Using autoencoders for session-based job recommendations." User Modeling and User-Adapted Interaction 30 (2020): 617-658. https://doi.org/10.1007/s11257-020-09269-1
10. Lars Ruthotto, and Eldad Haber. "An introduction to deep generative modeling." GAMM-Mitteilungen 44, no. 2 (2021): e202100008. https://doi.org/10.1002/gamm.202100008
11. Ivan Kobyzev, Simon JD Prince, and Marcus A. Brubaker. "Normalizing flows: An introduction and review of current methods." IEEE transactions on pattern analysis and machine intelligence 43, no. 11 (2020): 3964-3979. https://doi.org/10.1109/TPAMI.2020.2992934
12. Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. "Neural autoregressive flows." In International Conference on Machine Learning, Stockholm, Sweden, July 10-15, 2018, pp. 2078-2087.
13. Diederik P Kingma, and Max Welling. "Auto-encoding variational Bayes". 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014.
14. Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. "Embedding-based news recommendation for millions of users." In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, Halifax, Canada, August 13-17, 2017 pp. 1933-1942. https://doi.org/10.1145/3097983.3098108
15. Marlesson R. O. Santana, and Anderson Soares. "Hybrid Model with Time Modeling for Sequential Recommender Systems.", Proceedings of the Workshop on Web Tourism co-located with the 14th ACM International WSDM Conference (WSDM 2021), Jerusalem, Israel, March 12, 2021. https://ceur-ws.org/Vol-2855/challenge_short_9.pdf
16. Kai Deng, Jiajin Huang, and Jin Qin. "HybridGNN-SR: Combining unsupervised and supervised graph learning for session-based recommendation." In 2020 International Conference on Data Mining Workshops (ICDMW), pp. 136-143. IEEE, 2020. https://doi.org/10.1109/ICDMW51313.2020.00028
17. Noveen Sachdeva, Giuseppe Manco, Ettore Ritacco, and Vikram Pudi. "Sequential variational autoencoders for collaborative filtering." In *Proceedings of the twelfth ACM international conference on web search and data mining*, Melbourne, Australia, February 11-15, 2019 pp. 600-608. 2019. https://doi.org/10.1145/3289600.3291007
18. Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. "Neural attentive session-based recommendation." In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, Singapore, November 6-10, 2017 pp. 1419-1428. https://doi.org/10.1145/3132847.3132926
19. Meirui Wang, Pengjie Ren, Lei Mei, Zhumin Chen, Jun Ma, and Maarten De Rijke. "A collaborative session-based recommendation approach with parallel memory modules." In Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval, Paris, France, July 21-25, 2019, pp. 345-354. https://doi.org/10.1145/3331184.3331210
20. Taegwan Kang, Hwanhee Lee, Byeongjin Choe, and Kyomin Jung. "Entangled bidirectional encoder to autoregressive decoder for sequential recommendation." In Proceedings of the 44th

International ACM SIGIR Conference on Research and Development in Information Retrieval, Canada, July 11-15, 2021, pp. 1657-1661. https://doi.org/10.1145/3404835.3463016
21. Fajie Yuan, Xiangnan He, Haochuan Jiang, Guibing Guo, Jian Xiong, Zhezhao Xu, and Yilin Xiong. "Future data helps training: Modeling future contexts for session-based recommendation." In Proceedings of The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020 pp. 303-313. https://doi.org/10.1145/3366423.3380116
22. Tianan Liang, Yuhua Li, Ruixuan Li, Xiwu Gu, Olivier Habimana, and Yi Hu. "Personalizing session-based recommendation with dual attentive neural network." In *2019 International Joint Conference on Neural Networks (IJCNN)*, Budapest, Hungary, July 14-19, 2019, pp. 1-8. https://doi.org/10.1109/IJCNN.2019.8852185
23. Zhitao Wang, Chengyao Chen, Ke Zhang, Yu Lei, and Wenjie Li. "Variational recurrent model for session-based recommendation." In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Torino, Italy, October 22-26, 2018, pp. 1839-1842. https://doi.org/10.1145/3269206.3269302
24. Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer." In Proceedings of the 28th ACM international conference on information and knowledge management, Beijing. China November 3-7, 2019, pp. 1441-1450. https://doi.org/10.1145/3357384.3357895
25. Jianli Zhao, Hao Li, Lijun Qu, Qinzhi Zhang, Qiuxia Sun, Huan Huo, and Maoguo Gong. "DCFGAN: An adversarial deep reinforcement learning framework with improved negative sampling for session-based recommender systems." Information Sciences 596 (2022): 222-235. https://doi.org/10.1016/j.ins.2022.02.045
26. Rong Gao, Haifeng Xia, Jing Li, Donghua Liu, Shuai Chen, and Gang Chun. "DRCGR: Deep reinforcement learning framework incorporating CNN and GAN-based for interactive recommendation." In *2019 IEEE International Conference on Data Mining (ICDM)*, Beijing, China, November 8-11, Beijing, China, pp. 1048-1053. IEEE, 2019. https://doi.org/10.1109/ICDM.2019.00122
27. Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. "Generative adversarial user model for reinforcement learning based recommendation system." In *International Conference on Machine Learning*, Long Beach, California, USA, June 11-13, 2019, pp. 1052-1061.
28. Wei Zhao, Benyou Wang, Min Yang, Jianbo Ye, Zhou Zhao, Xiaojun Chen, and Ying Shen. "Leveraging long and short-term information in content-aware movie recommendation via adversarial training." *IEEE transactions on cybernetics* 50, no. 11 (2019): 4680-4693. https://doi.org/10.1109/TCYB.2019.2896766
29. Zhe Xie, Chengxuan Liu, Yichi Zhang, Hongtao Lu, Dong Wang, and Yue Ding. "Adversarial and contrastive variational autoencoder for sequential recommendation." In *Proceedings of the Web Conference 2021*, Ljubljana, Slovenia, April 19-23, 2021, pp. 449-459. https://doi.org/10.1145/3442381.3449873
30. Homanga Bharadhwaj, Homin Park, and Brian Y. Lim. "RecGAN: recurrent generative adversarial networks for recommendation systems." In *Proceedings of the 12th ACM Conference on Recommender Systems*, Vancouver, Canada, October 2, 2018, pp. 372-376. https://doi.org/10.1145/3240323.3240383
31. Yao Lv, Jiajie Xu, Rui Zhou, Junhua Fang, and Chengfei Liu. "SSRGAN: A Generative Adversarial Network for Streaming Sequential Recommendation." In Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part III 26, pp. 36-52. Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-73200-4_3
32. Wei Zhao, Benyou Wang, Jianbo Ye, Yongqiang Gao, Min Yang, and Xiaojun Chen. "Plastic: Prioritize long and short-term information in top-n recommendation using adversarial training." In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), Stockholm, Sweden, July 13-19, 2018, pp. 3676-3682. https://doi.org/10.24963/ijcai.2018/511

33. Ruiyang Ren, Zhaoyang Liu, Yaliang Li, Wayne Xin Zhao, Hui Wang, Bolin Ding, and Ji-Rong Wen. "Sequential recommendation with self-attentive multi-adversarial network." In Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, China, July 25-30, 2020, pp. 89-98. https://doi.org/10.1145/3397271.3401111
34. Duo Liu, Yang Sun, Xiaoyan Zhao, Gengxiang Zhang, Rui Liu. "Adversarial Training for Session-based Item Recommendations," 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2020, pp. 1162-1168, https://doi.org/10.1109/ITAIC49862.2020.9338819
35. Fan Zhou, Yuhua Mo, Goce Trajcevski, Kunpeng Zhang, Jin Wu, and Ting Zhong. "Recommendation via collaborative autoregressive flows." Neural Networks 126 (2020): 52-64. https://doi.org/10.1016/j.neunet.2020.03.010
36. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville and Yoshua Bengio. "Generative Adversarial Nets." NIPS (2014), Montréal, Canada, Dec 8-13, 2014, pp. 2672–2680.
37. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." Nature 323, no. 6088 (1986): 533-536. https://doi.org/10.1038/323533a0
38. Danilo Rezende, and Shakir Mohamed. "Variational inference with normalizing flows." In International conference on machine learning, Lille, France, July 6-11, 2015, pp. 1530-1538.
39. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. In Proceedings International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016. https://doi.org/10.48550/arXiv.1511.06939
40. Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. "STAMP: short-term attention/memory priority model for session-based recommendation." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, London, United Kingdom, August 19 - 23, 2018, pp. 1831-1839. https://doi.org/10.1145/3219819.3219950
41. Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. "Collaborative denoising auto-encoders for top-n recommender systems." In Proceedings of the ninth ACM international conference on web search and data mining, San Francisco, California, USA, February 22-25, 2016, pp. 153-162. https://doi.org/10.1145/2835776.2835837
42. Zhenlei Wang, Jingsen Zhang, Hongteng Xu, Xu Chen, Yongfeng Zhang, Wayne Xin Zhao, and Ji-Rong Wen. "Counterfactual data-augmented sequential recommendation." In Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval, Canada, July 11-15, 2021, pp. 347-356. https://doi.org/10.1145/3404835.3462855
43. Wang-Cheng Kang, and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In 2018 IEEE International Conference on Data Mining (ICDM).
44. Guijuan Zhang, Yang Liu, and Xiaoning Jin. "A survey of autoencoder-based recommender systems." *Frontiers of Computer Science* 14, no. 2 (2020): 430-450. https://doi.org/10.1007/s11704-018-8052-6
45. Anega Maheshwari, Priyanka Mitra, and Bhavna Sharma. "Autoencoder: Issues, Challenges and Future Prospect." In: Vashista, M., Manik, G., Verma, O.P., Bhardwaj, B. (eds) Recent Innovations in Mechanical Engineering. Lecture Notes in Mechanical Engineering. Springer, Singapore. https://doi.org/10.1007/978-981-16-9236-9_24
46. Min Gao, Junwei Zhang, Junliang Yu, Jundong Li, Junhao Wen, Qingyu Xiong, "Recommender systems based on generative adversarial networks: A problem-driven perspective", Information Sciences, Volume 546, 2021, Pages 1166-1185. https://doi.org/10.1016/j.ins.2020.09.013
47. Xiaopeng Li, James She. "Collaborative variational autoencoder for recommender systems", in: KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge

Discovery and Data Mining, Halifax, Canada, August 13-17, 2017, Pages 305–314. https://doi.org/10.1145/3097983.3098077
48. Yifan Chen, Maarten de Rijke. “A collective variational autoencoder for top-n recommendation with side information”, in: DLRS 2018: Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems, Vancouver, Canada, October 2018 October 2018, Pages 3–9. https://doi.org/10.1145/3270323.3270326
49. Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian personalized ranking from implicit feedback." In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal Quebec, Canada, June 18 - 21, 2009, pp. 452-461.
50. Ian, Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv:1701.00160, Dec. 2016. https://doi.org/10.48550/arXiv.1701.00160

Chapter 5
Hybrid/Advanced Session-Based Recommender Systems

Abstract The deep learning models in SBRS which have been discussed in the previous chapters have their own strengths and weaknesses. Due to the high flexibility of deep neural networks, many neural network blocks can be integrated to construct more robust and accurate models. Many session-based recommender system utilize hybrid deep neural network models. There are also several advanced deep learning approaches that are very popular in SBRS, including graph neural networks (GNNs) and deep reinforcement learning (DRL). To this end, advanced and hybrid deep neural network models in SBRS are discussed in this chapter.

Keywords Session-based recommender systems · SBRS · Graph neural network · Deep reinforcement learning · Hybrid models

5.1 Introduction

One of the main reasons for the popularity of deep learning methods is that they eliminate the need to manually perform feature extraction on unstructured data, which is challenging to do with traditional machine learning models. Since the understanding and detailed feature engineering of the dataset is critical to the final performance of the models, many algorithms based on deep learning achieve better results in this stage due to providing better feature extraction of deep models using more hidden layers than shallow models [1]. To achieve this level of accuracy, deep neural networks are much more efficient than other methods in terms of computation and the number of parameters. Deep neural networks can learn a deep and more abstract representation of the input in each layer [2]. Each deep learning method has specific features and capabilities, and due to the high flexibility of deep neural networks, many neural structure blocks can be integrated to formulate more robust models [3]. Hybrid deep learning models should follow the nature and characteristics of the problem scope and can be developed logically with high precision for specific purposes. The integration of two or more different models provides the possibility of using the advantages of each method, limiting their disadvantages, and strengthening the capabilities of the resulting integrated method.

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2_5

Different deep learning methods can be combined to deliver more accurate results in various research, but in the field of recommender systems, the combination of CNN and AEs, CNNs and RNNs, RNNS and AEs, etc. has received the most attention [3]. Many session-based recommender systems also utilized hybrid deep neural network models, which, due to the sequential nature of user interactions, mostly use the configuration whose important part includes RNNs. These approaches are discussed in Sect. 5.3 of this chapter.

In addition to hybrid deep neural network methods consisting of two or more types of basic deep neural networks, two other types of advanced approaches are very popular in session-based recommender system that are usually employed with other models: first, the approaches that utilize deep graph neural networks (GNNs) as the fundamental component, and second, the approaches that employ deep reinforcement learning (DRL) as the core module. These approaches in SBRS are discussed in Sects. 5.4 and 5.5 of this chapter, individually and in integration with other deep learning models.

This chapter of the book is organized as follows. First, a brief overview of the fundamental of these hybrid deep learning models, commonly used datasets, and evaluation baselines/metrics used in various research in this field will be discussed. Then, in Sect. 5.3, the hybrid deep neural network methods used in session-based recommender systems will be considered. In Sects. 5.4 and 5.5, the advanced approaches based on deep graph neural networks and deep reinforcement learning models are discussed and reviewed. Section 5.6 discusses and analyzes the results and the existing issues related to the hybrid/advanced deep learning models in SBRS and provides guidelines for future research in this scope.

5.2 Fundamentals

Each deep learning method discussed in detail in Chap. 2 of this book has its characteristics and advantages, which are used in different fields based on these characteristics. In most research related to session-based recommender system, multilayer neural networks are used to model non-linear interactions between items and users, convolutional neural networks are used to extract local and global representations from homogeneous data sources, and recurrent neural networks are used to model sequential or temporally ordered data [4]. In fact, single deep learning models are based on a single deep architecture, while hybrid deep models combine more than one single deep learning method through an effective communication technique [5]. Hybrid deep learning methods, in addition to benefiting from the advantages of single deep learning methods, also reduce the disadvantages of each method based on the capabilities that other models provide. In Table 5.1, single and hybrid deep learning models are compared from different aspects.

Because of the data complexity in SBRS, many models presented in this field are based on hybrid deep learning methods. In Chaps. 3 and 4 of the book, approaches using a single deep neural network were discussed, while in this chapter, the details

Table 5.1 Comparison of single and hybrid deep learning models

Single deep learning model	Hybrid deep learning model
Feature extraction is done for a limited scope	Feature extraction is done for a wider scope
Limited options for transfer learning	More options for transfer learning
Less effective than hybrid deep learning methods	Superior performance than single models
Fewer hardware resources required	High computational power
Less complicated than hybrid deep learning methods	The model complexity in both time and space is high
Limited applications	Various applications

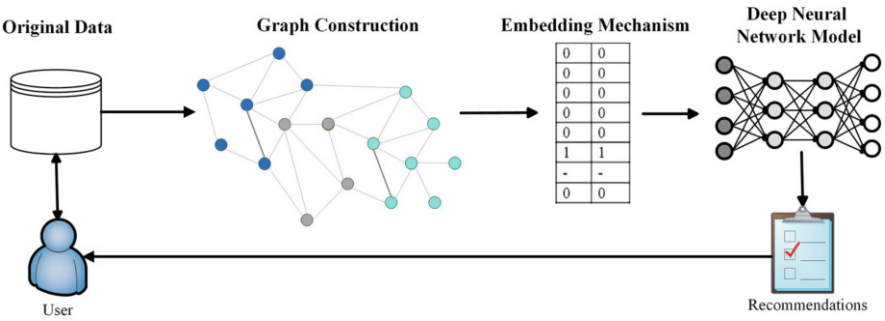

Fig. 5.1 The general architecture of graph neural network in recommender systems

of approaches based on a combination of deep learning models in SBRS are considered. In addition to hybrid deep neural network methods, deep graph neural networks and deep reinforcement learning are reviewed, both as a stand-alone approach and in integration with other models.

Although conventional deep learning techniques have achieved many successes in various fields, most of their data has been Euclidean, while many data are efficiently represented by graph structures [6]. Non-Euclidean data can more accurately represent complex concepts than one- or two-dimensional representations. One of the essential non-Euclidean structures is a graph. Graphs are a special type of data structure consisting of nodes connected by edges and can be used to model most problems in the scope of social networks. The need to cover non-Euclidean data in deep learning methods has led to using deep learning methods and graph concepts in various research fields.

One critical challenge of recommender systems is learning representations of items and users, and recently graph neural networks have been used to learn data representations very effectively [7, 8]. Figure 5.1 shows the general architecture of graph neural networks in recommender systems. It should be mentioned that the general framework of GNN in SBRS is presented in Sect. 5.4, Fig. 5.13.

Deep learning methods could learn accurate representations of graph-based data due to their ability to learn non-linear interactions between the users and the items. The combination of deep learning methods and graph neural networks has led to

many successes in various fields [9]. One of these areas is session-based recommender system. In SBRS, it is possible to model the sequential behaviors and user-item interactions with a graph and learn the relations between users and items using deep graph neural network. Moreover, graph neural networks could be combined with CNN and RNN models to provide more accurate and effective recommendations. The details of deep graph neural network methods and several approaches presented in this field have been considered in the third section of this chapter.

The reinforcement learning approach is focused on objective-oriented learning through interactions, where the learning agent determines the action that receives the most reward through the trial-and-error paradigm. This approach provides solutions that can model user-agent interactions. However, when faced with problems with high dimensions or continuous agents, inefficient representation of the feature and increasing learning time occur. To alleviate this issue, deep learning techniques are added to the reinforcement learning methods. They can automatically find compact and low-dimensional representations of high-dimensional data [10]. Therefore, combining the benefits of reinforcement learning and deep learning, deep reinforcement learning models are proposed, which are widely used in various fields. These types of models could learn the user's past interactions in different states and spaces.

In session-based recommender systems that dynamically recommend items to users, deep reinforcement learning methods are used to maximize expected long-term cumulative rewards. Such approaches can optimize recommendations for long-term user interactions instead of maintaining a short-term goal of optimizing the process of providing immediate recommendations to the user. Deep reinforcement learning methods enable recommender agents to learn optimal recommendation policies to recommend items to users [11]. The details of the deep reinforcement learning method and several approaches presented in this field have been considered in the fifth section of this chapter. Figure 5.2 shows the general architecture of using reinforcement learning in recommender systems. It should be mentioned that the general framework of DRL in SBRS is presented in Sect. 5.5, Fig. 5.23.

To summarize, the proposed research methods discussed in this chapter focus on three types of hybrid/advanced deep learning classifications, including hybrid deep neural networks, deep graph neural networks, and deep reinforcement learning. The first category, hybrid deep neural network methods, consists of the combination of CNN with GRU, CNN with LSTM, and autoencoders with RNNs. The second category, deep graph neural network methods, includes the methods based on GNNs and the composition of GNNs with GRU, LSTM, GAN, and CNN models. Finally, deep reinforcement learning methods include research using DRL or DRL with GRU and CNN. Table 5.2 presents the list of these studies according to the employed deep learning approach.

The diagram in Fig. 5.3 shows the percentage of each technique used in the discussed research.

According to the diagram in Fig. 5.3, a large percentage of the research reviewed is related to methods in which RNNs play a crucial role. This is inevitable because of the high capability of recurrent neural networks in processing sequential data related to session-based recommender systems. On the other hand, a quick review of the

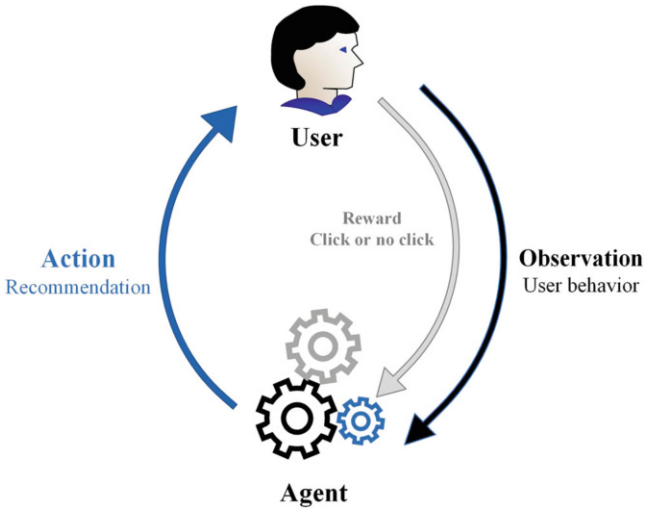

Fig. 5.2 The general architecture of reinforcement learning in recommender systems

Table 5.2 The list of research discussed in Chap. 5 using hybrid/advanced deep learning models

Deep learning model		References
Hybrid deep neural networks	CNN + GRU	[12–21]
	CNN + LSTM	[22–26]
	AE + RNN	[27–29]
Deep graph neural networks	GNN	[30–44]
	GNN + RNN	[45–52]
	GCN	[53–66]
Deep reinforcement learning	Deep Q-Learning	[10, 67–70]
	DRL + RNN	[71–74]
	DRL + CNN	[75, 76]
	DRL + GAN	[77–79]

listed research shows that many of them have used graph neural networks as the main component together with different combinations with other methods.

In hybrid deep neural network approaches, the largest percentage belongs to the combination of CNN and RNNs (GRU, LSTM) due to the potential of CNNs for automatic feature extraction and the capacity of RNNs for modeling sessions and user interactions.

In the following two subsections, a summary of the employed datasets and evaluation metrics of the reviewed research is presented.

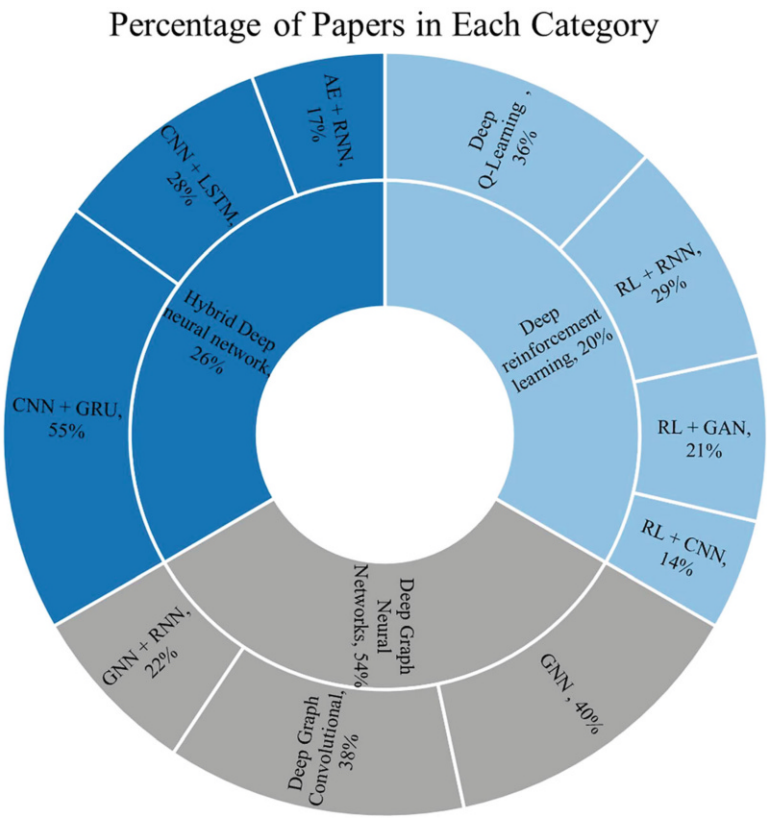

Fig. 5.3 Percentage of each type of hybrid/advanced deep learning models

5.2.1 Datasets

The evaluation of the proposed methods in each article is performed on datasets that contain different features, such as user interactions, items, event time and location, sessions, explicit user ratings, etc. Datasets are usually collected from various fields, such as e-commerce, news, movies, jobs, etc. To evaluate the proposed SBRS, one or more datasets are usually selected according to the intrinsic characteristics of the proposed method. Table 5.3 shows the datasets used in different articles, including the dataset name, the domain, a brief description, and the paper that employed it.

According to Table 5.3, some datasets, such as YooChoose, Diginetica, and Tmall, have been used in more research for evaluation. These three datasets are in the field of e-commerce, and their data are related to user interactions, which are fully compatible with the functional intrinsic of session-based recommender system. In SBRS, first, sessions must be captured in each dataset, and based on different factors, several sessions are considered for training and several others for model testing. For example, in the YooChoose dataset, which has many events, different

Table 5.3 Widely used datasets in SBRS using hybrid/advanced deep learning models

Dataset	Domain	Description	References
Diginetica	E-commerce	The dataset includes user sessions extracted from an e-commerce search engine log	[17, 20, 34, 36–39, 48–51, 54–61, 77]
YooChoose	E-commerce	The dataset consists of 6 months of clickstreams from an e-commerce Web site	[17–21, 26, 28, 34, 36–39, 48, 49, 51, 56, 60, 61]
Gowalla	POI	This dataset is from a location-based social networking Web site where users share their locations by checking in	[21, 25, 50, 51]
Last.fm	Music	This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system	[14, 19, 23, 31, 50, 51]
XING	Job posting	It is the XING RecSys Challenge 2016 dataset that contains interactions on job postings. User interactions come with timestamps and interaction types (click, bookmark, reply, and delete)	[15, 28, 31]
Reddit	News	It is on user activity on the social news aggregation and discussion Web site Reddit	[31]
TMall	E-commerce	This is the large dataset released in the IJCAI-15 challenge, which is collected from Tmall, the largest business-to-consumer e-commerce Web site in China. It records two types of user behaviors, views and purchases	[12, 18, 37, 45, 46, 54, 57, 59]
AOTM	Music	This dataset includes the user-contributed playlists from the Art of the Mix Web site	[12, 58]
JD.com	E-commerce	JD.com is one of the largest Chinese e-commerce Web sites that contains consumer purchasing behaviors, user ratings, reviews, and product metadata	[71, 75]
30MUSIC	Music	It is a collection of listening and playlists data retrieved from Internet radio stations through Last.fm API	[12, 45]
Nowplaying	Music	Nowplaying is created from music-related tweets, where users posted which tracks they were currently listening to	[34, 45, 54, 57]
MovieLens	Movie	It consists of users' sequential rating records for different categories of movies on the MovieLens site	[21, 27, 72]
Adressa	News	Adressa is a news dataset that contains reading behaviors and sessions from users	[13, 14, 23, 24, 47, 53]
MIND	News	The MIND is collected from Microsoft News. The news articles include title, abstract, body, category, and entities	[53]
Globo.com	News	Globo.com is the most popular media company in Brazil. The second version of	[13]

(continued)

Table 5.3 (continued)

Dataset	Domain	Description	References
		Globo.com also includes contextual information	
Tianchi	E-commerce	Tianchi is based on user-commodity behavior data from Alibaba's mobile commerce platforms	[46]
Foursquare	POI	This dataset contains check-ins in NYC and Tokyo collected for about 10 months. Each check-in is associated with its timestamp, its GPS coordinates, and its semantic meaning	[21, 25]
Retailrocket	E-commerce	The data has been collected from a real-world e-commerce Web site. It is raw data, i.e., without any content preprocessing; however, all values are hashed due to confidential issues	[18, 49, 58, 59]

sequences of events are sorted based on time, and after identifying the sessions, 1/4 or 1/64 of the total sequences are obtained according to the proposed method. Accordingly, two versions of YooChoose 1/4 and YooChoose 1/64 have been used to evaluate the proposed methods [17]. In the Adressa dataset, there are session start and end tags, and sessions are identified this way, but in datasets such as Last.fm, time intervals are used to identify sessions. For example, [14] considered the time interval of a session to be 30 min, and if two events happened less than 30 min apart, they were placed in one session; otherwise, they were placed in two different sessions.

In some research, in addition to determining sessions, mechanisms are also used to determine how to use sessions to train the model and evaluate it. For example, in [13], after identifying the sessions, all the sessions that occurred in 1 h are divided into a group and sorted based on time. The sessions related to each hour are placed in a group, and every 5 h are used to train the model to evaluate and predict the sixth hour. Then, the sessions from the beginning to the 10th session are used for training the model, and the 11th session is tested. This process continued until the end.

Several research, such as [34, 36, 37, 54, 57], which employed Diginetica or Nowplaying datasets, split and labeled sessions before using them. In the process of splitting, the generated sequence contains the first item and the label of the second item. The next sequence contains the first and second items and the label of the third item. This procedure goes to the point that if a session contains n items, its last sequence contains $n-1$ items and the label of the n^{th} item. Depending on the model, some methods based on graph neural networks utilize different special preprocessing on datasets. For example, in [45], users participating in fewer than two sessions are removed to ensure that each user graph contains interactions with enough previous sessions.

Some session-based recommender systems that use the reinforcement learning method (such as [10]), in addition to the offline and online evaluation using some

datasets, have also utilized the RecSim simulation platform. RecSim is a configurable platform and simulation environment for recommender system that support sequential user interactions. This platform enables to create new environments that reflect specific aspects of user behavior and item structure at a level of abstraction that is well suited for applying reinforcement learning constraints and recommender system techniques to sequential interactive recommendation problems.

Table 5.4 presents the information on each dataset, including the number of sessions/items/events, duration of data collection, average length of the session, type of interaction, and access link to the dataset.

5.2.2 Evaluation

Generally, to evaluate recommender systems, the data of the selected dataset is divided into two parts, train and test, so that the simulated proposed model is trained based on the training data and evaluated using the test data. To evaluate session-based recommender systems, the proper dataset must first be selected, and the data divided into different session. For splitting at the event level, all events before a certain time can be considered for training and the remaining events for testing. However, in splitting at the session level, the time of the first event of a session is considered, and based on that, it is determined that this session is related to the training or test set. Both approaches are useful, but using the second approach makes it possible for all user interactions in one session to be in the training or the test set. But, in the first case, part of the interactions may be included in the test set and another part in the training set [80]. The performance of each proposed method is quantified based on different evaluation metrics, and the obtained values are compared with the results of the baseline recommender systems. The most widely used baseline methods commonly used to evaluate all session-based recommender systems are listed below. It should be noted that some of them have been introduced as baselines in the previous chapter. However, to keep the comprehensiveness and consistency of the content of this chapter, they are also repeated in this section:

- **POP:** More popular items are always recommended. The POP is effective and straightforward simultaneously and is often a strong baseline in specific domains [81].
- **S-POP:** The most popular items in the current session are recommended. The recommendation list changes based on the number of events that are related to particular items. This baseline is useful for the domains with high repetitiveness [81].
- **Ppop:** Based on the Ppop method, items are recommended that a user interacts mostly [12].
- **Item-KNN:** Items similar to the actual item are recommended, and the similarity between them is measured based on the cosine similarity measure of their session

Table 5.4 Characteristics of widely used datasets

Domain	Dataset	Number of sessions	Number of items	Number of events	Timespan	Average session length	Interaction type	Access link
E-commerce	YooChoose	7,981,581	37,486	31,708,505	182 days	3.97	Click/buy	https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015
	Diginetica	204,789	43,136	993,483	–	4.85	Click	https://cikm2016.cs.iupui.edu/cikm-cup
	JD.com	1,000,000	93,243	5,524,491	3 years and a quarter	–	Click/buy	https://wise.cs.rutgers.edu/dataset/
	Retailrocket	75,405	55,050	191,197	135 days	3.54	Click/buy	https://www.kaggle.com/retailrocket/ecommerce-dataset
	TMall	1.77M	425,348	13.42M	91 days	7.56	Click/buy	https://ijcai-15.org/index.php/repeat-buyers-prediction-competition
Video	MovieLens ML30	858,160	18,273	25,368,155	17 years	–	Rating (1–5)	http://files.grouplens.org/datasets/movielens/
	MovieLens ML100	300,624	18,226	25,240,741	17 years	–	Rating (1–5)	http://files.grouplens.org/datasets/movielens/
News	Adressa	982,210	13,820	2,648,999	16 days	2.7	Click	http://reclab.idi.ntnu.no/dataset
	Globo.com (G1)	1,048,594	46,033	2,988,181	16 days	2.84	Click	https://www.kaggle.com/datasets/gspmoreira/news-portal-user-interactions-by-globocom
	Reddit	1,135,488	27,452	3,848,330	30 days	3	Click	https://www.kaggle.com/colemaclean/subreddit-interactions
POI	Gowalla	830,893	29,510	1,122,788	240 days	3.85	Check-in	https://snap.stanford.edu/data/loc-gowalla.html
Music	Last.fm	169,576	449,037	2,887,349	95 days	17.03	Click	http://millionsongdataset.com/lastfm/
	AOTM	21,888	91,166	306,830	95 days	14.02	Click	http://www.artofthemix.org
	30MUSIC	152,189	138,274	1,995,778	365 days	11	Play	http://recsys.deib.polimi.it/?page_id=54
	Nowplaying	126,249	30,673	976,702	95 days	7.73	Click	http://dbis-nowplaying.uibk.ac.at/#nowplaying

vectors. In other words, it is the number of co-occurrences of two items in sessions divided by the square root of the product of the number of sessions in which the individual items occurred. This method is very effective for evaluating item-to-item recommendation methods [82].

- **GRU4Rec:** A technique based on recurrent neural networks, which is one of the first approaches to using deep learning techniques in session-based recommender system. This method is based on GRU and is used to overcome the problem of gradient vanishing [81].
- **NARM:** An improved version of GRU4Rec, which performs session modeling by introducing a hybrid encoder based on the attention mechanism. In this technique, global and local encoders are defined, the global encoder corresponds to the GRU4Rec method, and the local encoder is proposed for adding the attention mechanism to the model, respectively [83].
- **STAMP:** This method is based on a Short-Term Attention/Memory Priority Model and, unlike the NARM method, is not based on a recurrent neural network. In this method, users' general interests are obtained through the long-term memory data of the session context, and their short-term interests are also recognized through short-term memory [84].
- **BPR-MF:** It utilizes matrix factorization which is optimized for pairwise ranking objective functions through stochastic gradient descent. Methods based on matrix factorization cannot be used in session-based models because there is no pre-computed feature vector for new sessions. This problem is overcome by using the average vectors of the items that belong to each session [85].

In addition to the above methods, some baseline methods are based on graph neural networks and are used to evaluate graph-based recommender systems. Some of the most commonly used ones are:

- **SR-GNN:** This method provides a session-based recommender system using a graph-based neural network that creates latent vectors of items and presents each session through an attention-based network [36].
- **FGNN:** It learns the vectors of items through a weighted attention graph layer and learns the features of sessions through the feature extraction layer of a session graph [33].
- **A-PGNN:** This method converts all sessions of a user into a graph and uses a gated graph neural network to learn item transitions. It also uses an attention mechanism to explicitly model the effects of the user's previous interests on the current session [86].

In the evaluation process of recommendation systems based on reinforcement learning, some of the most widely used baseline methods are as follows:

- **DQN:** This method is the result of combining deep learning and reinforcement learning, which uses a deep neural network to estimate the Q function. DQN is used for scenarios that have a discrete action space [87].

- **DEERS:** The main idea of this method is based on using the user's negative feedback in addition to his positive feedback. Q network has two inputs for positive and negative states [71].
- **DDQN:** It is based on the DQN method, which decomposes its objective function, choosing the optimal action and calculating the target Q value from each other. It uses a dual network structure for action selection and value evaluation [88].

In this section, a number of evaluation metrics that are used more in this field have been discussed in the following:

- *Mean Reciprocal Rank (MRR):* MRR focuses on the rank of relevant items in the list of recommendations. It shows that placing a relevant item at the top of the recommendation list significantly impacts user satisfaction and is calculated using Eq. (5.1):

$$\text{MRR@}N = \frac{1}{Q}\sum_{i=1}^{Q}\begin{cases}\dfrac{1}{\text{rank}_i} & \text{if } \text{rank}_i \le N \\ 0 & \text{otherwise}\end{cases} \tag{5.1}$$

  where Q is a sample of recommendation lists and rank_i refers to the rank position of the relevant item for the i-th recommendation list.

- *Recall:* This metric is calculated based on the number of relevant items that are among the top N items in the recommendation list, and the rank of the relevant items in the N list is unimportant, and it is calculated using Eq. (5.2):

$$\text{Recall@}N = \frac{\text{Number of relevant items in top } N \text{ list}}{\text{Total of relevant items}} \tag{5.2}$$

- *Precision @ N*: This metric evaluates the number of relevant items relative to the total N items recommended in the list, and it is calculated using Eq. (5.3):

$$\text{Precision@}N = \frac{\text{Number of relevant items in top } N \text{ list}}{\text{Total of } N \text{ items}} \tag{5.3}$$

- *Coverage@N:* It checks the coverage of the items. Item coverage measures the percentage of items that are ever recommended, and the variety of the recommended items in the recommendation list is considered. Its goal is to recommend a high percentage of various items to the user. This metric is calculated using Eq. (5.4):

$$\text{Coverage@}N = \frac{\text{Distinct items that appeared in any top} - N \text{ recommendation}}{\text{All distinct recommendable items}} \tag{5.4}$$

- *Hit Rate@N*: It is the percentage of times in which relevant items are retrieved among the top N ranked items, and it is calculated using Eq. (5.5):

$$\text{Hit Rate@}N = \frac{1}{Q}\sum_{i=1}^{Q}\begin{cases} 1 & \text{if } \text{rank}_i \leq N \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

where Q is a sample of recommendation lists and rank_i refers to the rank position of the relevant item for the i-th recommendation list.

- *$nDCG_p$:* This metric is based on cumulative gain (CG). The cumulative gain is the sum of the graded relevance values of all items in a recommendation list. nDCG is computed as the ratio between discounted cumulative gain (DCG) and idealized discounted cumulative gain (IDCG). Equations (5.6), (5.7), and (5.8) show how to calculate this measure.

$$\text{DCG}p = \sum_{i=1}^{p}\left(\frac{2^{r_i}-1}{\log_2(i+1)}\right) \tag{5.6}$$

$$\text{IDCG}p = \sum_{i=1}^{\text{REL}p}\left(\frac{r_i}{\log_2(i+1)}\right) \tag{5.7}$$

$$n\text{DCG}p = \frac{\text{DCG}p}{\text{IDCG}p} \tag{5.8}$$

In the above equations, r_i is the graded relevance of the result at position i, and RELp represents the list of *relevant* items (ordered by their relevance) up to position p.

- *MAP:* This metric calculates the average precision. In fact, after each relevant item is recommended, the precision is measured, and the average is calculated using Eq. (5.9):

$$\text{MAP} = \frac{\sum_{q=1}^{Q}\text{Ave}P(q)}{Q} \tag{5.9}$$

In this relation, *P(q)* is the precision of query q, and parameter Q is the number of queries.

- *Mean Absolute Error (MAE):* This metric is one of the most common errors of prediction factors, which calculates the mean absolute value of the difference between the score predicted by the system and the actual score of the item. The mean absolute error indicates the degree of closeness of the recommendations to reality. This measure can be calculated from Eq. (5.10):

$$\mathrm{MAE} = \left(\frac{1}{N} \sum_{i \in O_u} \left| P_{u,i} - r_{u,i} \right| \right) \tag{5.10}$$

- *Root Mean Square Error (RMSE):* The metric of the root mean square error of the predicted rank is more effective than the mean absolute error in problems where the errors are more considerable, and it is calculated using Eq. (5.11):

$$\mathrm{RMSE} = \sqrt{\frac{1}{N} \sum_{i \in O_u} \left(P_{u,i} - r_{u,i} \right)^2} \tag{5.11}$$

In Eqs. (5.10) and (5.11), $P_{u,i}$ is the predicted score for the item i by user u, $r_{u,i}$ is the actual value of the score assigned to item i by user u, O_u is the set of items rated by user u, and N is the total number of predictions made by the system.

- *Area Under the ROC Curve (AUC):* Another important metric used to determine the efficiency of recommender systems is the AUC. The larger the AUC value, the more favorable the final system performance is evaluated. The ROC (receiver operating characteristic) space is formed by two indices FPR on the horizontal axis and TPR on the vertical axis, as calculated by Eqs. (5.12) and (5.13), respectively. The line that connects two points (0,0) and (1,1) divides the ROC space into two parts. The area above this line is the favorable area and below the line is the unfavorable area. Therefore, the AUC is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5.12}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{5.13}$$

Apart from the above evaluation metrics that are generally used in many methods, some measures are specifically used in the reviewed approaches in this chapter:

- *ESI-R@N:* This metric *expresses* the expected self-information with rank sensitivity, which is used to indicate the degree of novelty of the recommendations, and it is calculated using Eq. (5.14):

$$\mathrm{ESI}-R(L)=\frac{1}{\sum_{j=1}^{N}\mathrm{disc}(j)}\sum_{k=1}^{N}-\log_2 p(i_k)*\mathrm{disc}(k) \tag{5.14}$$

The main term of the above equation is $\log_2 p(i)$, which originates from the concept of self-information and indicates the amount of information conveyed through the observation of an event. The log function is used to emphasize the impact of new items. $p(i)$ is the probability that each item is part of a random user interaction representing the recent normalized popularity of that item. $\mathrm{disc}(k)$ is a logarithmic scaling function that maximizes the novelty effect of items at the top of the suggestion list. This function is shown in Eq. (5.15):

$$\mathrm{disc}\ (k)=\frac{1}{\log_2 k+1} \tag{5.15}$$

- *ESI-RR@N:* This metric is based on $\mathrm{ESI}-R(L)$ and expresses the expected self-information with rank and relevance sensitivity, and it is calculated using Eq. (5.16):

$$\mathrm{ESI}-\mathrm{RR}(L)=C_k\sum_{k=1}^{N}-\log_2 p(i_k)*\mathrm{disc}(k)*\mathrm{relevance}\ (i_k,u) \tag{5.16}$$

the relevance (. , .) in the above equation indicates the degree of relevance. If the desired item is among the list of items that the user interacts with in the current session, its relevance value is 1. Otherwise, its value is equivalent to the background probability for unobserved items that the user does not interact with but are somewhat relevant to him. C_k is calculated using Eq. (5.17):

$$C\ (k)=\frac{1}{\sum_{\acute{K}=1}^{N}\mathrm{disc}\left(\acute{k}\right)} \tag{5.17}$$

- *EILD-R@N:* This metric is used to calculate the amount of diversity in recommendations and actually shows the expected intra-list diversity with rank sensitivity, and it is calculated using Eq. (5.18):

$$\text{EILD} - R(L) = \frac{1}{\sum_{\acute{k}=1}^{N} \text{disc}(\acute{k})} \sum_{k=1}^{N} \text{disc}(k) \frac{1}{\sum_{\acute{l}=1:\acute{l}\neq k}^{N} \text{rdisc}(\acute{l}, k)} \sum_{\acute{l}=1:l\neq k}^{N} d(i_k, i_l) * \text{rdisc}(l, k) \tag{5.18}$$

In the above equation, $\text{rdisc}(l, k)$, which is calculated according to Eq. (5.19), represents a relative ranking discount, which considers item l that is ranked before the target item k has already been discovered.

$$\text{rdisc}(l, k) = \text{disc}(\max(0, l - k)) \tag{5.19}$$

- *EILD-RR@N*: This metric is based on *EILD-R@N* and is sensitive to relevance in addition to rank, and it is calculated using Eq. (5.20):

$$\text{ILD} - \text{RR}(L) = C_k \sum_{k=1}^{N} \text{disc}(k) * \text{relevance}\ (i_k, u) C_l \sum_{l=1:l\neq k}^{N} d(i_k, i_l) * \text{rdisc}(k, l) * \text{relevance}\ (i_l, u) \tag{5.20}$$

In the above equation, C_k is a normalization term which is a weighted average based on rank discounts, and it is calculated using Eq. (5.21):

$$C_l = \frac{1}{\sum_{\acute{l}=1:\acute{l}\neq k}^{N} \text{rdisc}(k, \acute{l})} \tag{5.21}$$

Table 5.5 shows the different evaluation metrics used in different articles on session-based recommender systems using hybrid/advanced deep learning models.

5.3 SBRS Using Hybrid Deep Neural Networks

Before looking at the approaches of hybrid deep neural network models in SBRS, it is appropriate to discuss why this type of integration can be helpful in achieving more accurate results in this scope. Then, we fully discuss different combinations of deep neural network models for this purpose.

Table 5.5 Widely used evaluation metrics in SBRS using hybrid/advanced deep learning models

Evaluation metrics	References
Mean reciprocal rank (MRR)	[12–15, 17–23, 31, 34, 36–38, 45, 46, 48–51, 53–57, 59–61]
Recall@n	[12, 14, 15, 18–23, 25, 27–29, 45, 46, 49, 56, 58, 61, 75]
Mean rank percentile (MRP)	[15]
nDCG	[16, 58, 68, 69, 71, 72, 75, 76]
Click-through rate (CTR)	[10, 69]
AUC	[24, 54]
Precision@n	[14, 17, 21, 25, 28, 34, 36–38, 48, 51, 54, 57, 59, 60, 69, 75, 77]
Hit rate	[13, 16, 25, 31, 50, 72]
F1	[14, 24, 26, 47, 75]
RMSE	[27, 29, 77]
MAP	[25, 29, 68, 71, 75, 76]
Coverage	[10, 13]
EILD-R@n	[13, 53]
EILD-RR@n	[13, 53]
ESI-R@n	[13, 53]
ESI-RR@n	[13, 53]

5.3.1 Why Hybrid Deep Neural Network?

Hybrid deep neural network methods are used in various fields to recognize more accurate features and provide a more optimal model. Because of the sequential nature of the data, session-based recommender systems usually use recurrent neural networks to model the sequence of events in this way. On the other hand, for more accurate feature extraction, achieving more optimal representations of inputs, and obtaining better results, other deep learning methods can be combined with recurrent neural networks. The research reviewed in this section shows that most hybrid deep neural network methods in the field of SBRS have used a combination of CNN and GRU, CNN and LSTM, and AE and RNN. A higher percentage of these studies are based on GRU since LSTMs have more gates/parameters, which causes a higher computational complexity. Each of these three categories will be discussed in the following sections.

5.3.2 Approaches Based on CNN and LSTM

Many studies of a session-based recommender system are based on hybrid deep neural network methods and usually include a type of recurrent neural network such as GRU and LSTM. In this subsection, methods based on the combination of LSTM and CNN are discussed.

Park et al. have proposed a session-based news recommender system based on the combination of two methods, RNN and CNN, which uses an LSTM as a sub-type of

RNNs [22]. They used a personalized re-ranking approach that considers user interests based on a weighted average of categories of articles the user reads. This method estimates categories using the classification based on convolutional neural networks for data that does not have category information. The embedding vector related to each news article employing the bag-of-words representation of queries, titles, and content of the news has been trained. In fact, each news article is converted into a vector based on the Doc2Vec method. These vectors are then sent as input to LSTM to create user representations based on their clicked news representation vectors. In this article, two types of recurrent neural network approaches are investigated: history-based and session-based. In the history-based approach, all user clicks are considered, while in the session-based approach, only clicks from the current session are considered. The session-based approach has obtained better results that show the impact of users' short-term interests in predicting their next clicks in session-based recommender systems.

A Deep Joint Network (DeepJoNN) that uses a combination of deep learning methods is proposed by Zhang et al. [23]. DeepJoNN utilized a combination of three-dimensional CNN and recurrent neural networks. LSTM in DeepJoNN optimizes the traditional RNN by integrating a memory cell vector during each session to address the problem of exploding/vanishing gradient when learning long-term dependencies. The features that are considered as input for each news article include the article ID, keywords, entities, and category to which the relevant news belongs, and character-level embedding is used for their embedding. Character-level embedding has benefits of lower computation complexity and fewer parameters. The only limitations appear in the processing of misspellings and informal words, which rarely happen in the news field. 3D CNN is used to process input data and extract news features. These types of CNNS are tensor-based and use 3D filters for feature extraction.

The session-parallel mini-batch method is used in DeepJoNN to increase efficiency in the model training, which is suitable for models in which different users have different sessions with different numbers of interactions. The inputs are considered a tensor or 3D array with dimensions: number of features ×length of feature embedding × mini-batch size. LSTM is also used to detect sequential patterns between user clicks and their related features. In addition, DeepJoNN presents a time decay function to calculate the freshness of news articles within a time period; beyond this time period, the news article will hardly attract the interest of users.

The architecture of DeepJoNN is shown in Fig. 5.4. As shown in this figure, DeepJoNN consists of two types of deep neural networks, which are coupled hierarchically and thus can extract textual patterns and process long-term and short-term dependencies simultaneously.

Zhu et al. proposed a Deep Attention Neural network (DAN) model for a session-based news recommendation system, which consists of three components: parallel CNN (PCNN), attention-based neural network (ANN), and attention-based recurrent neural network (ARNN) [24]. The ARNN is used to find the sequential features of users' clicked news, and the ANN is used to detect the features of users' current interests with respect to the candidate news. The ARNN works based on LSTM. ANN designs an attention-based neural network that automatically matches each

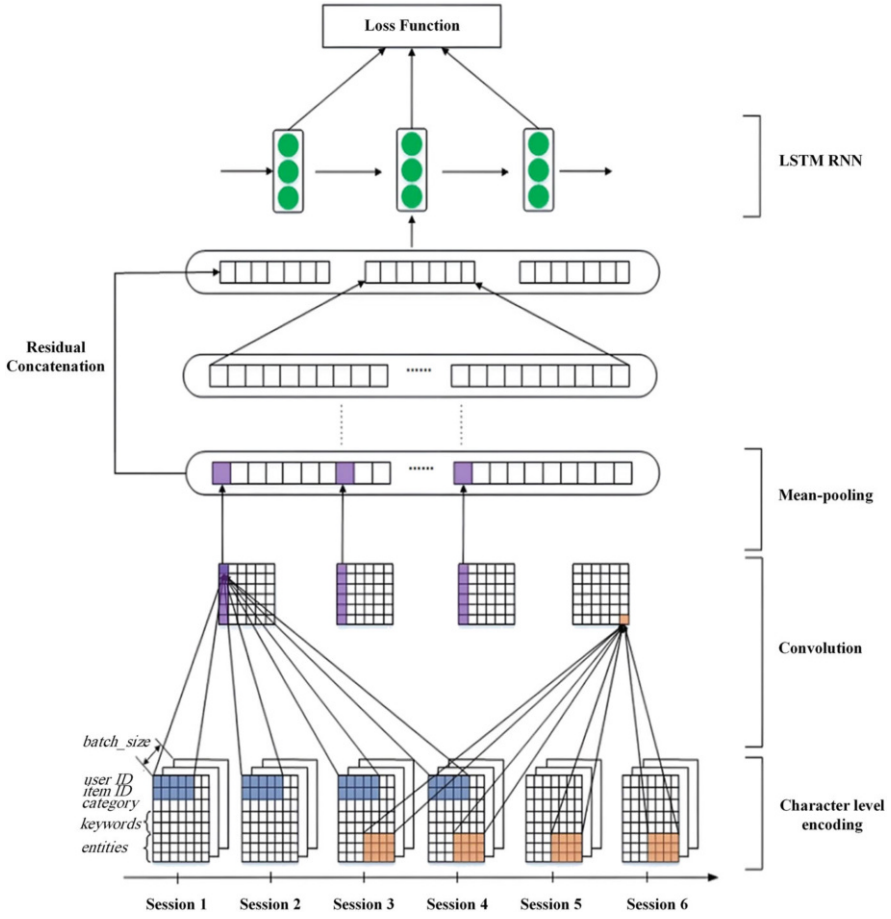

Fig. 5.4 The architecture of DeepJoNN [23]

clicked news with candidate news and aggregates the user's current interests with different weights. The PCNN component uses two parallel CNNS to extract news presentations based on the news titles and news profiles. These two parallel CNNS combine the information related to the title as well as the entities related to the news text (profile).

To learn user features, not only ANN is used to model the user's current interests, but also the ARNN component is used to obtain potential sequential features of a user's history readings. Therefore, in DAN, the sequential features of the user's click history and the user's current preferences together provide the user's preferences and features. Finally, DAN recommends the news to the user according to the similarity between the user feature representation and the candidate news representation. The architecture of DAN is shown in Fig. 5.5.

Due to the success of RNNs and CNNs in detecting local sequential patterns and complex long-term dependencies between data, Xu et al. proposed a model based on

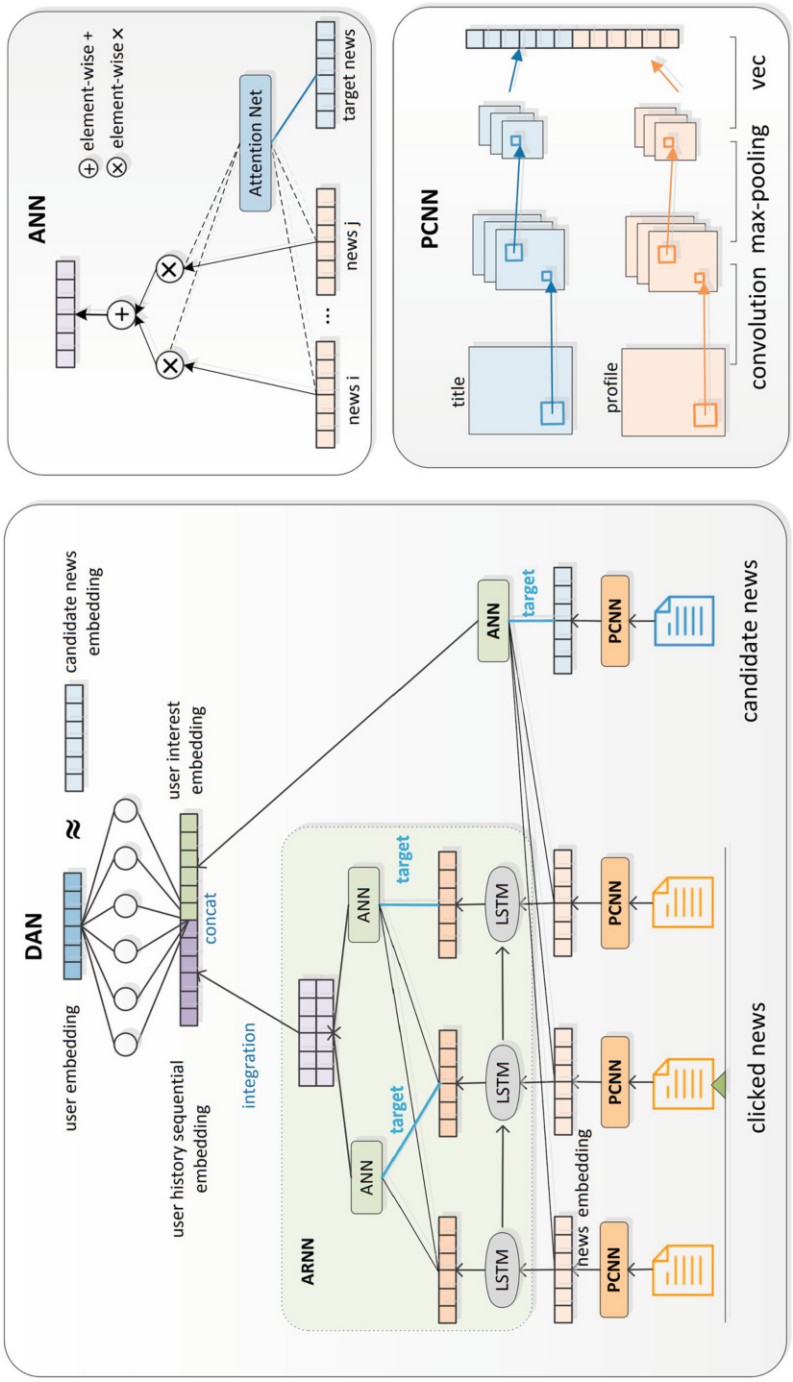

Fig. 5.5 The architecture of DAN [24]

a recurrent convolutional neural network (RCNN) [25]. RCNN combines the short-term and long-term interests of the user to provide a high-level hybrid representation for sequential recommender systems. LSTM has been utilized in RCNN to detect long-term dependencies and CNN to extract local sequential patterns among hidden states. RCNN, when the current item enters the recurrent layer, creates a hidden state at each time step, which is a hidden representation of the user's sequential interests. Then, the recent hidden states are considered as images in each time step, and their local sequential features are investigated and searched using vertical and horizontal convolutional filters. An intra-step horizontal filter is used to detect non-linear features of interactions, while a vertical inter-stage filter is used to detect non-monotone local patterns. The output of the convolutional neural network and the LSTM hidden state vector are combined to describe the user's overall interest. It is then fed into a fully connected layer to predict the list of recommendations. The architecture of RCNN is shown in Fig. 5.6.

In some research, deep learning methods are combined with some classical machine learning methods. Bedi et al. [26] have combined CNN and LSTM with fuzzy time series to recommend products to users based on their activities performed in a session, called FS-CNN-LSTM-SR [26]. The FS-CNN-LSTM-SR considers the numerical and categorical features of session-based recommender systems. To this end, fuzzy logic controls the uncertainty in the user's interests, convolutional layers extract the important features, and multivariate LSTM learns the sequential dependencies between input data and learns patterns of fuzzy and non-fuzzy features. In fact, LSTM is suitable for data with long sequential dependencies but takes a long time to train. To solve this issue in FS-CNN-LSTM-SR, LSTM has been placed next to CNN to reduce the number of epochs needed for training. FS-CNN-LSTM-SR includes two general phases: preprocessing and recommendation. The preprocessing consists of filling the empty values, retrieving required fields, extracting new features, converting original time series data into fuzzy time series, data normalization, and padding. The recommendation phase consists of two modules: prediction and recommendation. In the prediction module, the items that qualify for the recommendation are predicted, and in the recommendation module, a set of recommended items is selected and specified for the user.

5.3.3 Approaches Based on CNN and GRU

Approaches based on recurrent neural networks usually have low speeds and are difficult to train for high-volume data due to the gradient backpropagation problem. CNN-based methods also have high memory consumption, and latent representations are not interpretable. Therefore, some SBRS use hybrid deep neural network methods based on GRU and CNN, which can properly use their strengths and reduce the limitations of each. In this subsection, the proposed research methods based on the combination of CNN and GRU are discussed.

Due to the large volume of interactions, items, and users in SBRS, single deep learning models such as RNNs or CNNs cannot capture the hierarchical interests of

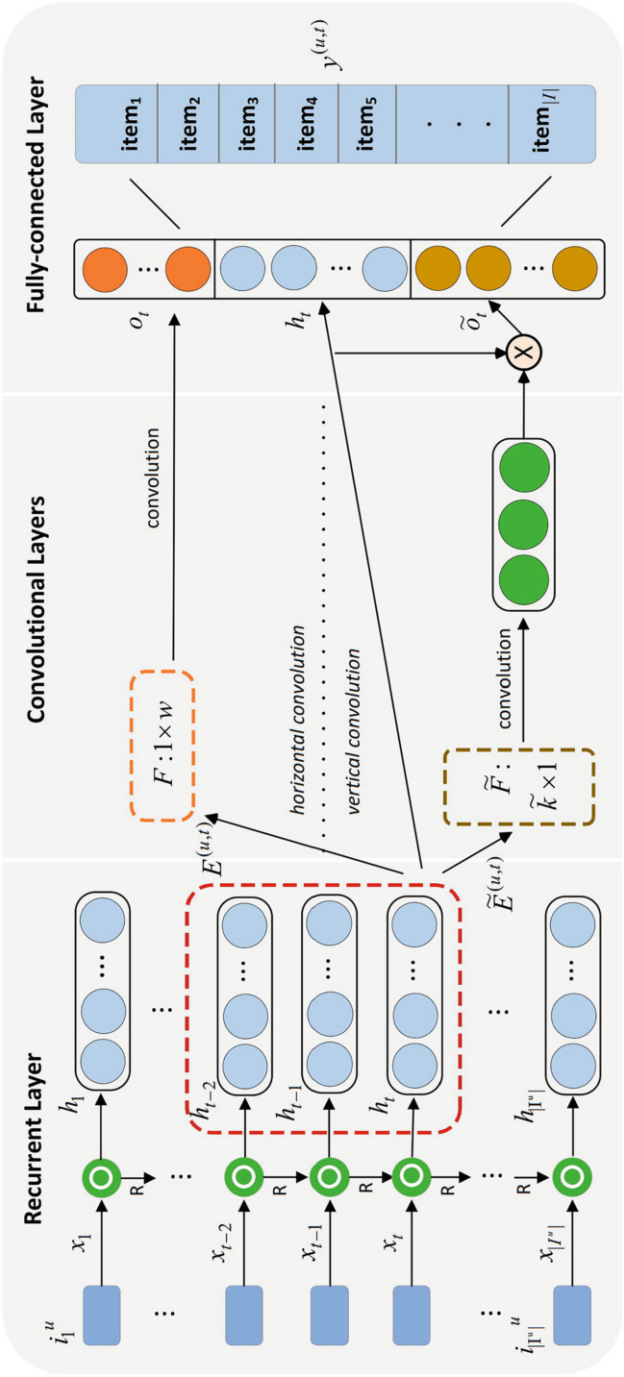

Fig. 5.6 The architecture of RCNN [25]

users in a session or between different sessions. Therefore, You et al. have proposed a Hierarchical Temporal Convolutional Networks (HierTCN) architecture for modeling the sequential interactions of users in large-scale and real-time recommender systems [15]. HierTCN consists of two levels that can recognize the hierarchy of user interests. The high-level model uses a recurrent neural network to detect long-term user interests between sessions. The output of the low-level model generates a dynamic user embedding that is created based on the user's long-term interests and short-term interactions within sessions. In HierTCN, the high-level model is implemented based on GRU, and the low-level model is implemented based on TCN (temporal convolutional networks). Users' long-term interests are represented using the GRU hidden state. Hidden states are updated through the integration of item embedding vectors in each session. The low-level model presents the user's short-term interests and predicts the user's next interaction in the session. This model is based on TCN, a special type of convolutional neural network with sequential inputs. Its structure includes causal convolution, dilated convolution, residual block, and residual connections. The combination of these powerful components leads to the achievement of acceptable efficiency and performance for TCN. Using causal convolution and dilated convolution in TCN increases the speed of calculations, and, similar to recurrent neural networks, TCN accepts inputs with variable lengths. It is worth mentioning that TCN's input is the user's long-term interests and its output is the user's dynamic embedding.

HierTCN uses a two-phase mechanism to update user interests and item embedding. Users' interests change rapidly, so the embedding vectors of users' interests should be updated in real time, but the properties of items change less quickly. In the first phase of the updating mechanism, the item embedding vectors are created based on convolutional graph neural networks. In this graph, each node represents an item, and each edge is drawn between nodes that interact with the corresponding items quickly. For each embedding item, the structural features of the graph are also used in addition to the textual and image features. In the second phase, the embedding vectors of the items are considered fixed, and the user's interest vectors are created based on them. The general architecture HierTCN is shown in Fig. 5.7.

Moreira et al. developed a meta-architecture, called CHAMELEON, that includes some structural blocks for news recommender systems that can be set up in different ways based on the characteristics of the target system [13]. The core of this architecture consists of two modules whose inference and training processes are independent of each other [89]. The Article Content Representation (ACR) module is used to learn a distributed presentation of the content of articles, and the Next Article Recommendation (NAR) module is used to recommend future articles in current sessions of users. The ACR module comprises the submodules of Textual Features Representation (TFR) and Content Embeddings Training (CET). In TFR, the text content of news articles is processed using CNN and sent to fully connected layers. The CET submodule can also learn, in a supervised or unsupervised manner, the vectors for presenting articles generated by TFR. The ACR module learns the embedding of the contents of an article independently of the records of users' sessions. This is done for reasons of higher scalability because if the number of

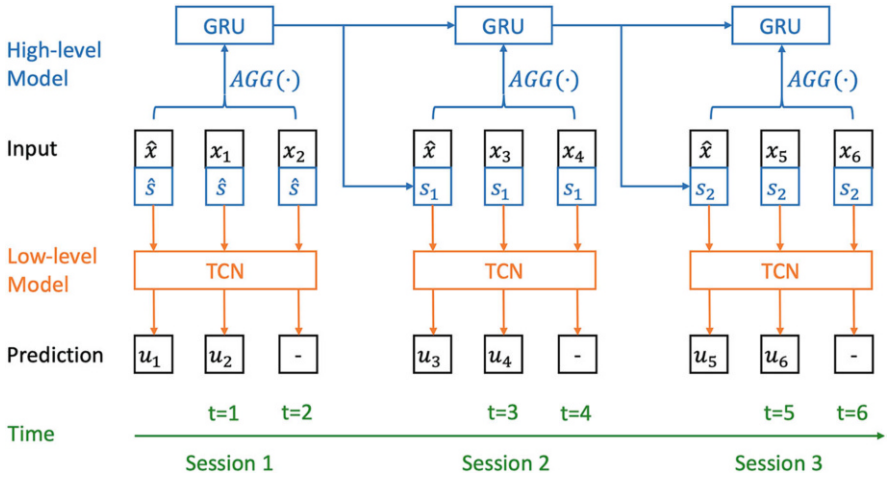

Fig. 5.7 The architecture of the HierTCN [15]

interactions between users and news articles is huge, the process of learning these interactions is computationally expensive in a joint process.

Then the content vectors of the learned articles are stored in a memory and fed as input to the NAR module. The NAR provides recommendations for active sessions based on previous user interactions and news article content. NAR is designed to be context-aware and receives contextual information such as location, the device used by the user, and the previous user's clicks as well as information about the context of the articles, such as their popularity and recency as input. NAR consists of three submodules: Contextual Article Representation (CAR), SEssion Representation (SER), and Recommendation Ranking (RR).

The CAR combines NAR inputs, which include text embedding vectors of articles, contextual features of articles (recency and popularity), and contextual features of users. This component can be realized as a fully connected multilayer or factorization machine. The SER uses GRU recurrent neural networks to model users' sequential clicks. RR is also developed to maximize the similarity between the predicted embedding vector for the next article and the contextual personalized embedding vector of the article already read by the user in his session (a positive example), while it should minimize the similarity of the predicted embedding vector for the next article with negative examples (articles that the user does not read during the session). The architecture of CHAMELEON framework is shown in Fig. 5.8. It is worth mentioning that another instantiation of the CHAMELEON has been proposed by Moreira et al., in which the ACR module employs a CNN, and in the NAR module, the sequence of clicks from users' sessions is modeled by LSTM [90].

To take advantage of the recurrent neural network and CNN methods, Bach et al. proposed a SBRS that is based on the recurrent convolutional recommendation model, RecConRec [18]. RecConRec has two main features: (1) It contains a

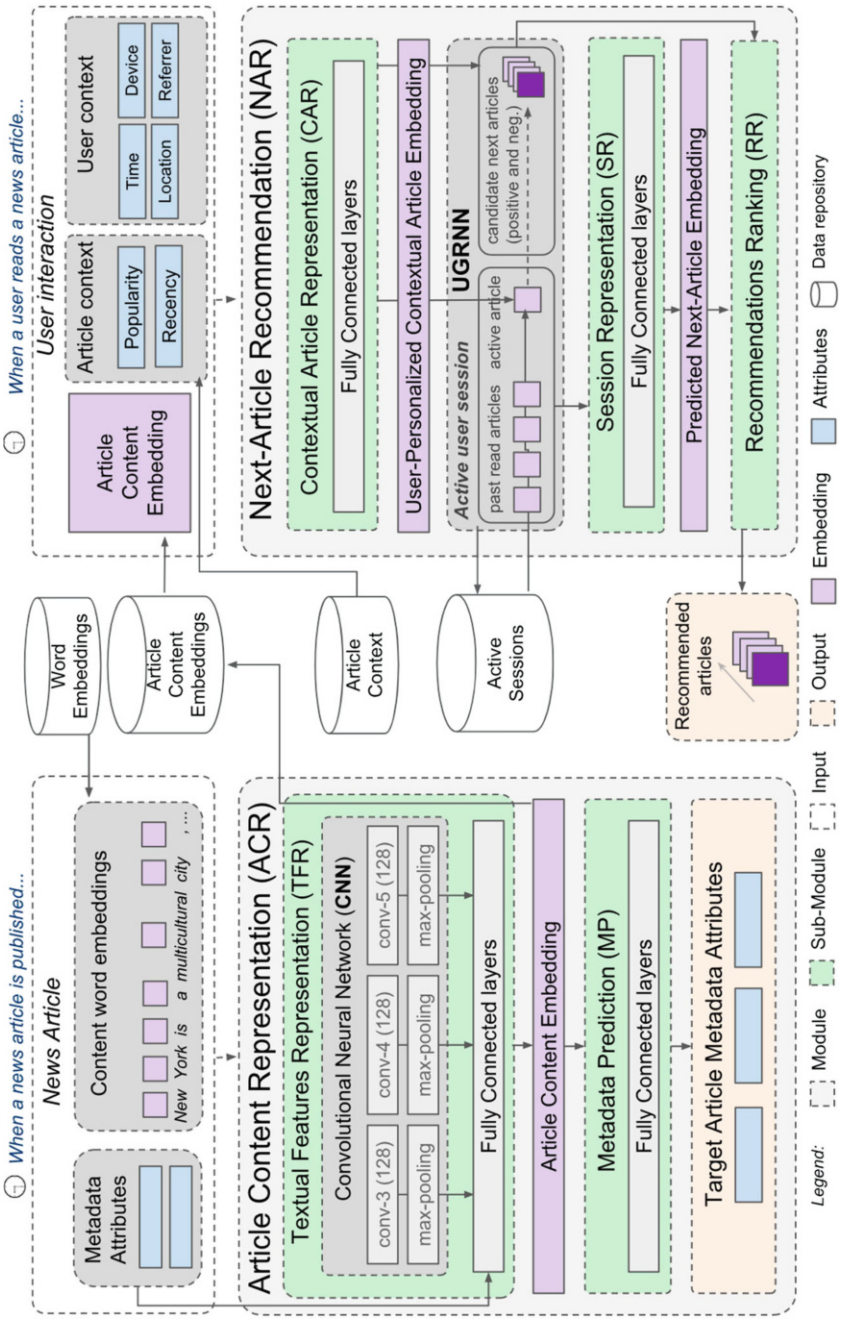

Fig. 5.8 The architecture of the CHAMELEON framework [13]

CNN layer that extracts the individual and collective associations inherent in a sequence of sessions. Individual dependency refers to the condition that each previous action of the user affects his next action, and on the other hand, collective dependencies indicate that several consecutive actions jointly affect the next user's action. To achieve these complex dependencies, RecConRec uses CNN iteratively on small windows of consecutive items in a session. By using convolutional filters of different sizes, this layer can extract individual and collective patterns from the data. (2) The GRU layer is used to capture long-term dependencies, and the input of the GRU is the output of the CNN layer. The effect of this layer is for the time when the previous actions do not affect the immediate next actions, but they affect the more distant ones. In fact, RecConRec is a combination of the CNN layer, which works on the items' embedding vectors and extracts complex local patterns, and the GRU layer, which recognizes long-term sequential patterns.

For item embeddings that are fed to the CNN layer, the one-hot vector of each item is created, and each vector is mapped to a space with smaller dimensions. Reducing the sizes of the items embedding vectors reduces the complexity of convolutional operations, and on the other hand, the embedding vectors can be trained together with the model so that the items that are seen together in the same session have similar embeddings. In RecConRec, the highway network layer between CNN and GRU is used to increase the effectiveness, which includes a multilayer perceptron network with residual connections.

Zhang et al. also combined CNN and GRU for SBRS [19]. In this method, Recurrent and Convolutional Neural Network for Session-based Recommendation (RCNN-SR), the user's general interest is determined based on the user's main purpose, the user's current interest is determined based on the user's sequential interactions and behaviors, and the user's dynamic interest is formed based on the changes in the user's choices and actions. In the first step, the user's general interest is extracted by a GRU with an item-level attention mechanism to dynamically select meaningful items in a session. Then, in the second step, the user's dynamic interest is detected through feature interactions and the user's current interests through item transitions. For this purpose, a vertical convolutional filter and multiple horizontal convolutional filters are used to extract the effects of non-monotonic sequential patterns and multivariate features. Finally, users' general, current, and dynamic preferences are concatenated and sent to a fully connected layer to provide recommendations.

RCNN-SR includes three encoders for the main purpose, dynamic preference, and sequential behaviors. The main-purpose encoder models the user's current interests. The input of this encoder is all the clicked items in the session, and its output is the user's current interest based on the candidate items clicked in the current session. This encoder is based on GRU, and the final hidden state, which summarizes the sequential behavior, is taken as the output of the current session. The input of the dynamic interest encoder is the embedding vectors of the clicked items in a session, and its output is the non-monotonic relationships between all features of the clicked items in the current session. This encoder uses a vertical filter for the vertical convolution operation to learn the item-specific features from the GRU

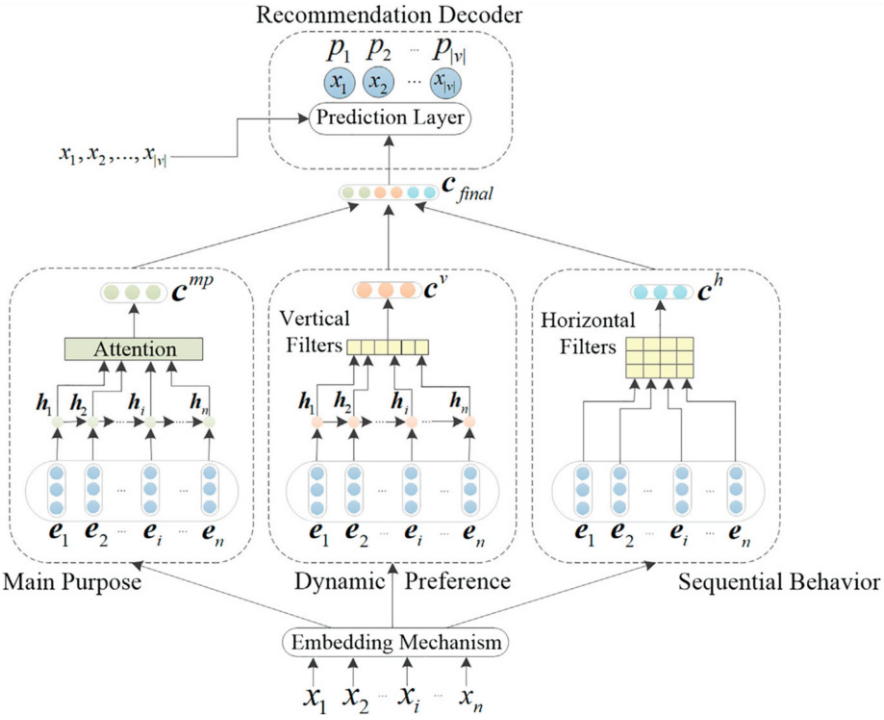

Fig. 5.9 The architecture of RCNN-SR [19]

hidden states. In the sequential behavior encoder, the input is the clicked items in a session, and the output is the non-linear transitions of the clicked items in the current session. Although user behaviors are sequential, there may not necessarily be dependencies between neighboring items, so the ability to detect point-level dependencies alongside union-level dependencies allows CNN to model high-level relationships. This method can reduce the problem of data sparsity. The architecture of the RCNN-SR is shown in Fig. 5.9.

To learn dependencies and general sequential patterns in session-based recommender system, Zhao et al. utilized the recurrent residual convolution networks to extract patterns from sequences of sessions [21]. The authors proposed a hybrid neural model, SGPD, for learning Sequential General Pattern and Dependency. SGPD includes four layers, the embedding, residual convolution, Bi-GRU, and user preference layers. In the embedding layer, item IDs are encoded in a continuous space with low dimensions to reduce the complexity of calculations and data processing. The next layer is the optimized recurrent residual convolution layer, where each recurrent residual block consists of a set of normalization and convolution operations and plays a key role in this method. In this layer, a sliding window is designed first, and the consecutive items in the window create the item block in the session. This window moves over items and learns general patterns of sequences of

interactions within the session. Finally, the output of this layer is sent to a fully connected neural network to convert to a feature vector.

SGPD provides a Bi-GRU model that scans the sequence forward and backward in the Bi-GRU layer to learn potentially personalized information about a user. In the user's preference layer, according to the results of the forward and backward scanning of the sequence, a representation of the user's feature vector is created. Finally, the features of the sequences are obtained by high-level abstraction through a fully connected neural network, and based on that, the representation of the session is determined. Accordingly, the score of each item is estimated for selection by the user as the next item.

5.3.4 Approaches Based on RNN and Autoencoder

Another important type of hybrid deep neural network used in SBRS is the combination of autoencoders and recurrent neural networks. Autoencoders are usually used to extract efficient representations for user interactions and feature transformations, and recurrent neural networks recognize sequential dependencies and long-term interests of the user. In this subsection, the proposed research methods based on AE and RNN are discussed.

Chen et al. proposed a sequential recommender system to predict the user's next transaction that employs an autoencoder on raw transaction data and submits observed transaction encodings to a GRU-based sequential model [29]. Their model, SEQNBT (SEQuential recommendation model for Next Best Transaction), predicts the user's next most likely transaction. Assuming that each user has different transactions with different industries, this method receives as input a sequence of previous transactions of the user and predicts the code of the industry category and the amount of the user's next transaction. SEQNBT consists of an autoencoder, GRU, and transaction decoder. Autoencoder is a self-training model that aims to encode transactions. Stacked autoencoders are used to extract efficient representations for each transaction. Extensive feature conversion is performed in this section. The characteristics related to the user's behavior, such as the number of transactions and mean/median/total transaction amount, and user's spending behavior, such as the reservation amount, i.e., the maximum amount that the user is willing to pay for a product, are integrated into different industry categories. Embedding descriptions of the types of industries in which the user has transactions are based on the BERT (Bidirectional Encoder Representation from Transformers) model. All the integrated features of the transactions, along with the transaction amount and embedding of the relevant industry category, are merged and sent to the encoder to be reconstructed by the decoder later.

Next, multiple GRU layers recognize sequential dependencies and the long-term interests of the user. In addition to the embedding vectors of transactions, the input of this layer includes time features so that the next transaction can be predicted using them. Once the model is trained, it is evaluated on an out-of-sample dataset to predict

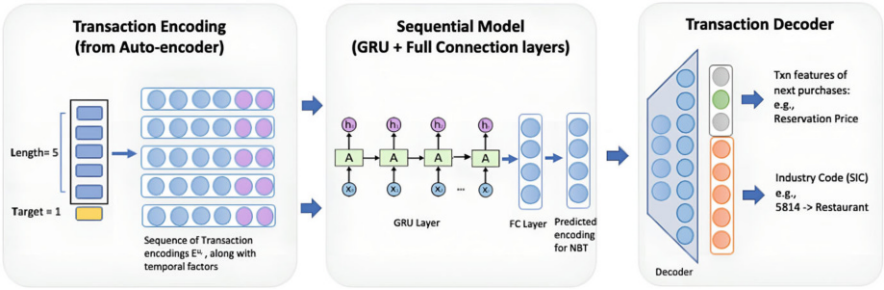

Fig. 5.10 The architecture of SEQNBT [29]

encodings for each user's next predicted transaction. The predicted transaction encryption is divided into two parts in the transaction decoding stage: the concatenated transaction features vector and the industry category embedding vector. Then, the cosine similarity between the predicted industry category embedding vector and all industry category embedding vectors in the feature table that store the BERT representations is calculated. The predicted transaction amount can be extracted from the features of the decoded transaction. The architecture of SEQNBT is shown in Fig. 5.10.

Although in most of the recent research, the time intervals have been considered as an explicit component and used in the learning process of sessions, usually, the effects of multiple previous levels are not considered in the proposed research; instead, a sequence with a limited length is considered. To this end, Fuentes et al. used deep learning models for the sequential prediction problem as a multi-class classification based on LSTM [28]. This method is based on the combination of LSTM, encoder, and decoder methods and automatically learns behavioral patterns from previous purchase transactions to predict the next purchase item or the category to which the next purchase item belongs. The architecture of this proposed method consists of four parts: transaction data, customer sequences file, LSTM training model input, and training model output. These components form the inputs and outputs of the three stages of the proposed method, which are converting customers' purchase sequences, creating multi-level favorites, and learning to buy favorites. An LSTM layer is capable of learning temporal dependencies, but a chain of LSTMs is more suitable for processing time-based sequential data; for this reason, a combination of encoder-decoder and stacked LSTMs is used in this method. The LSTM encoder processes user interests as input and generates an encoded representation. LSTM decoder uses decoder representation to generate output. A new customer presentation method is presented as the basis of the data transformation process, which allows working with multi-level interactions in scenarios where the length of sequences may be short and interactions have more dependencies on previous sessions.

To overcome the problem of data sparsity in the insurance industry, Borg Bruun et al. employed a combination of an autoencoder and recurrent neural networks to utilize the user's past sessions as signals to learn recommendations [27]. In

particular, this model learns several types of user actions that are not necessarily always related to items, and unlike other models of session-based recommender system, they model the relationships between input sessions and user target operations that are not performed in input sessions. The purpose of providing this recommender system based on cross-sessions is to recommend the next items that the user will buy according to the user's past sessions. This system has three distinct features: the target operation, which is purchasing, occurs outside the session, the user's operations are monitored in multiple sessions, and predicting what items the user will buy after the last time step.

Generally, in session-based recommender systems that use recurrent neural networks, the input of recurrent neural networks is the sequential items that the user has interacted with in a session and the output for each time step is the probability of selecting each item as an item that the user will interact with. This method, which is an extension of the GRU4REC method, takes multiple sessions of each user as input and predicts the items that the user will buy after the last time step by considering all types of user operations. Each session is assumed to be a sequence of user operations, and for each operation, a part of the Web site with which the user has interacted, an entity of the Web site selected by the user, and the user's interaction method with that entity must be considered.

Three methods are proposed for passing input sessions to recurrent neural networks. In the first method, which is cross-sessions encoding, a session is encoded by integrating session operations with the max pooling operator, and for each time step in user sessions, a GRU layer calculates hidden states. The second method is cross-sessions concat, where all sessions of a user are concatenated and form one session. A global order is considered for user operations, whereas the first method only considers the order of sessions. In both methods, model learning is performed through a multi-label classification. In the third method, cross-sessions auto, sessions are encoded automatically using an autoencoder. An autoencoder based on recurrent neural networks is trained with a GRU layer that takes as input a sequence of operations ordered in a session.

5.4 SBRS Using Deep Graph Neural Network

Before looking at the approaches of deep graph neural network models in session-based recommender systems, an overview of the GNN and the reasons that made it an effective choice for SBRS are provided.

5.4.1 Why Graph Neural Network?

Graph topology/structure encodes a large amount of information that is difficult to capture using traditional learning techniques. GNNs focus on learning mechanisms

that use this knowledge to achieve better performance for downstream tasks such as ranking prediction and similar content retrieval. In this regard, the goal of GNNs is to learn better representation/embedding of nodes using neighborhood information. GNNs can also be used to learn edges and graph representations.

Graph neural networks are a class of deep learning methods specifically developed to infer data described by graphs. GNNs can be applied directly to graphs to provide an optimal way to perform tasks such as node-level, edge-level, and graph-level prediction. Graph neural networks allow the end-to-end machine learning model that is simultaneously trained to learn a representation of graph-structured data. Graph neural networks can be applied to data with a graph structure for different purposes and can also learn representations at the node, edge, or graph level.

There are various approaches for training machine learning models on data with a graph structure using preprocessing techniques. However, they lack the flexibility to fully adapt to existing datasets and operations in machine learning models due to the high dimensionality and non-Euclidean intrinsic of graph data. It should be noted that the basic types of neural networks can only be implemented using regular or Euclidean data. However, almost all real-world data have a non-Euclidean dynamic graph structure. The irregularity of the many graph-structured data and the required parallel and scalable processing have led to recent advances in graph neural networks.

A graph-structured data is widely used in various fields such as image processing, recommender systems, social network analysis, etc. A graph is a data structure with two components, nodes and edges. Graph G is shown as $G=(V, E)$, while V is a set of nodes, and E is a set of edges that connect these nodes. $v_i \in V$ represents a node, and $e_{ij} = (v_i, v_j) \in E$ represents the edge between two nodes v_i and v_j. In general, graphs are classified as below [7]:

- Directed and undirected graphs: In directed graphs, edges connect a source node to a destination node, but in undirected graphs, edges only show the connection between two nodes.
- Homogeneous and heterogeneous graph: A homogeneous graph consists of one type of node and edges, but a heterogeneous graph consists of various kinds of nodes and edges.
- Hypergraph: A generalization of a graph in which an edge can connect any number of nodes.

The main idea of GNN is to iteratively summarize the feature information of the neighbors with respect to the graph data and integrate the collected information with the current central node representation during the propagation process [91, 92]. From the network architecture point of view, GNN stacks several propagation layers, which include aggregate and update operations. The propagation process was based on Eqs. (5.22) and (5.23):

$$\textbf{\textit{Update}}: \quad h_v^{l+1} = U\left(h_v^l, n_v^l\right) = U\left(h_v^l W^l + \sum_{u \in N(v)} \left(\frac{1}{c_{vu}}\right) h_u^l W^{l\prime}\right) \left(\tag{5.22}$$

$$\textbf{\textit{Aggregation}}: \quad n_v^l = A\left(\left\{h_u^l, \forall u \in N(v)\right\}\right) \tag{5.23}$$

In the above equations, h_u^l represents the node u in the l^{th} layer, n_v^l is the result of summarizing the aggregation function on neighboring nodes v in the l^{th} layer, A and U represent the aggregation and update operation functions in the l^{th} layer, W^l and $W^{l'}$ represent the learnable transformation matrices for the l^{th} layer, and N is the set of neighbors of the associated node.

It should be noted that in the aggregation stage, the existing works are mainly based on mean-pooling operation for each neighbor equally [93] or differentiating the importance of different neighbors by using the attention mechanism [94]. In the update phase, the representation of the central node and the aggregated neighborhood are merged into the updated representation of the central node.

Based on the concepts of graph neural networks, many deep learning models such as graph convolutional networks (GCN) [95], graph attention networks (GAT) [94], and different configurations of graph neural networks with other models such as RNNs (e.g., GGNN: Gated Graph Sequence Neural Networks [96]) have been developed and used in various research. GCN approximates the first-order eigendecomposition of the graph Laplacian to iteratively aggregate information from neighbors and uses a non-linear activation function, such as ReLU, for updating phase. GAT assumes that the influence of neighbors is neither identical nor pre-determined by the graph structure; therefore, using the attention mechanism, it distinguishes the participation of neighbors. In GGNN, the recurrent function is executed several times on all nodes in the aggregation phase and adopts a GRU model for updating phase. In addition to graph neural networks, GCNs have also been used in session-based recommender systems in various articles.

GCNs are suitable neural network architectures for machine learning on graphs. A GCN can generate significant feature representations and use its structural information in networks. GCNs use the concept of CNNs and define them for the open graph domain. A significant difference between CNNs and GCNs is that CNNs are specifically designed to work on regularly structured (Euclidean) data, while GCN is a generalized version of CNNs where the number of nodes' connections is different and the nodes are also irregular (irregular on non-Euclidean structured data).

If you are familiar with convolution layers in CNNs, "convolution" in GCNs is basically the same operation that refers to multiplying the input neurons by a set of weights, commonly known as a filter or kernel. GCNs perform similar operations to CNNs, where the model learns related features by considering neighboring nodes.

Researchers have classified GCNs into two types:

- Spatial GCN: This model of GCNs uses spatial features to learn from graphs that are located in spatial space. In many cases, graphs are coupled with spatial information embedded in their nodes. Standard GCNs do not consider the

location of nodes. These approaches directly perform convolution in the graph domain by collecting the information of neighboring nodes.

- Spectral GCN: This model of GCNs uses a special Laplacian graph decomposition to propagate information along the nodes. These networks are inspired by the propagation of waves in signals and systems study. The spectral graph convolution operation in the Fourier domain is performed by calculating the eigen decomposition of the graph Laplacian. The eigenvalue of the (normalized) graph Laplacian can be easily calculated from the adjacency matrix of the symmetric normalized graph. However, filters must be defined in Fourier space, and computing the Fourier transform of the graph is expensive (it requires multiplying the node features by the eigenvector matrix of the graph Laplacian, which is an $O(N^2)$ operation for a graph with N nodes).

Although spectral-based methods have more computational cost than spatial-based, they provide a stronger capacity to extract features from graph data.

In GCNs, the adjacency matrix A should be considered in addition to the node features (or the so-called input features) in the forward propagation equation. A is a matrix that represents edges or connections between nodes in the forward propagation equation. A describes the representation of the graph structure in the form of an adjacency matrix that enables the model to learn the representation of the features based on the connectivity of nodes. By adding the adjacency matrix as an additional element to the forward propagation representation of the features in neural networks, the forward propagation is calculated using Eq. (5.24):

$$h_v^{l+1} = \sigma\left(h_v^l W^l A^* + b^l\right) \tag{5.24}$$

σ is a non-linear activation function such as ReLU, b^l is a bias parameter in the l^{th} layer, W^l is the weight matrix in the l^{th} layer, h_v^l is a representation of the node v in the l^{th} layer, and A^* is the normalized version of A.

According to the above descriptions, three main steps should be considered for the development of GCNs:

1. Kernel/Filters: A filter is a function that acts like a scanner that has a limit on the number of cells (adjacency matrices) that must be considered.
2. Pooling: Similar to the function of the filter scanner, pooling is a function that outputs all the values scanned by the scanner at the same time. This output value can be calculated as the maximum, average, etc., of values. Only one output cell is generated after applying the kernel, followed by the pooling function.
3. Flattening: The flattening function reduces the network structure to a vector with lower dimensions, which can be used for the input of feedforward neural networks.

The above three steps are common to all GCNs. The only major difference is in the different kernel functions throughout the graph neural network. The general architecture of GCNs is shown in Fig. 5.11.

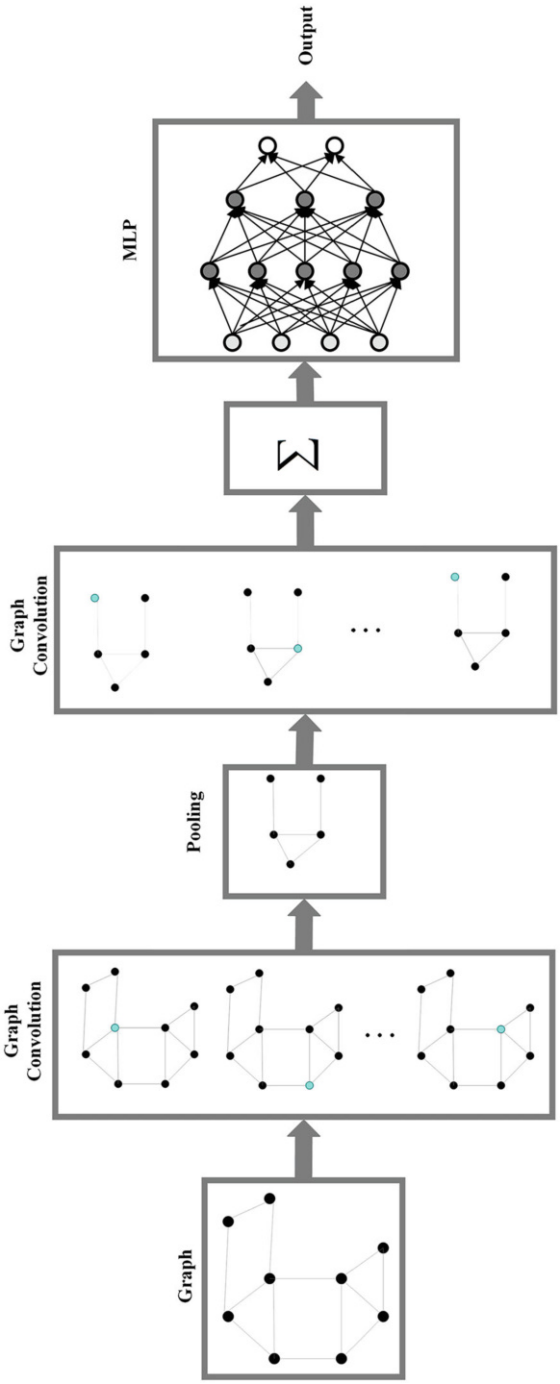

Fig. 5.11 The general architecture of GCNs

Different types of techniques based on graph neural networks have been proposed in various domains, and one of these domains belongs to recommender systems. Many recommender systems form a graph-structured data, so graph neural networks can be widely used due to their high capability in representation learning for graph data. For example, user-item interactions can be represented by a bipartite graph between the user and item nodes, where the edge represents the interaction between the user and the corresponding item. In addition, a sequence of items can be transformed into a sequence graph, so that each item can be connected to one or more subsequent items, and an edge is considered between consecutive items. Figure 5.12 shows an example of a bipartite and a sequence graph.

On the other hand, due to their high flexibility, graph neural networks provide the possibility that side information can also be easily modeled and used in addition to the main data. For example, social network data can be added to the user session dataset. In addition, a graph neural network can explicitly encode the important collaborative signal of user-item interactions to improve its representation through the propagation process. Compared with non-graph models, graph neural networks are more flexible and convenient for modeling multi-hop connectivity related to user-item interactions.

In session-based recommender systems, sequences of items can be modeled as a graph-structured data to represent adjacency between items. Graph neural networks are widely used to detect the transition pattern from the sequential behaviors of users by transforming them into a sequential graph. Figure 5.13 presents the general framework of graph neural networks in recommender system with sequential data.

In short, the use of GNNs in session-based recommender systems can be summarized by two general principles:

- **Graph construction**: In contrast to user-item interactions, which essentially have a bipartite graph structure, sequential behaviors are naturally expressed in time order. Therefore, constructing a sequence graph based on the sequential behavior of users is necessary to use GNN in session-based recommender system. In several systems, such as [33, 36, 54], the session data is modeled with a directed graph to capture the item transfer pattern. Of course, the sequence of sessions in session-based recommender systems is short, and user behaviors are limited; for example, the average length of sequences in Tmall is only 6.69 [54, 57], and in YooChoose, it is about 5.71 [36]. Therefore, a session graph constructed based on a single session may contain only a limited number of nodes and edges. To address the above challenge and capture possible relationships between items, there are two strategies: (1) directly capturing relationships from other sessions and (2) adding additional edges to the session graph.
- **Aggregation of neighbor information**: Given a constructed sequence graph, it is essential to design an effective propagation mechanism to capture transition patterns among items. In particular, the propagation process can be defined in the simplest possible form and as a mean pooling to aggregate previous and subsequent items. A combination of two aggregated representations and using GRU to integrate the information of neighbors and the central node are other

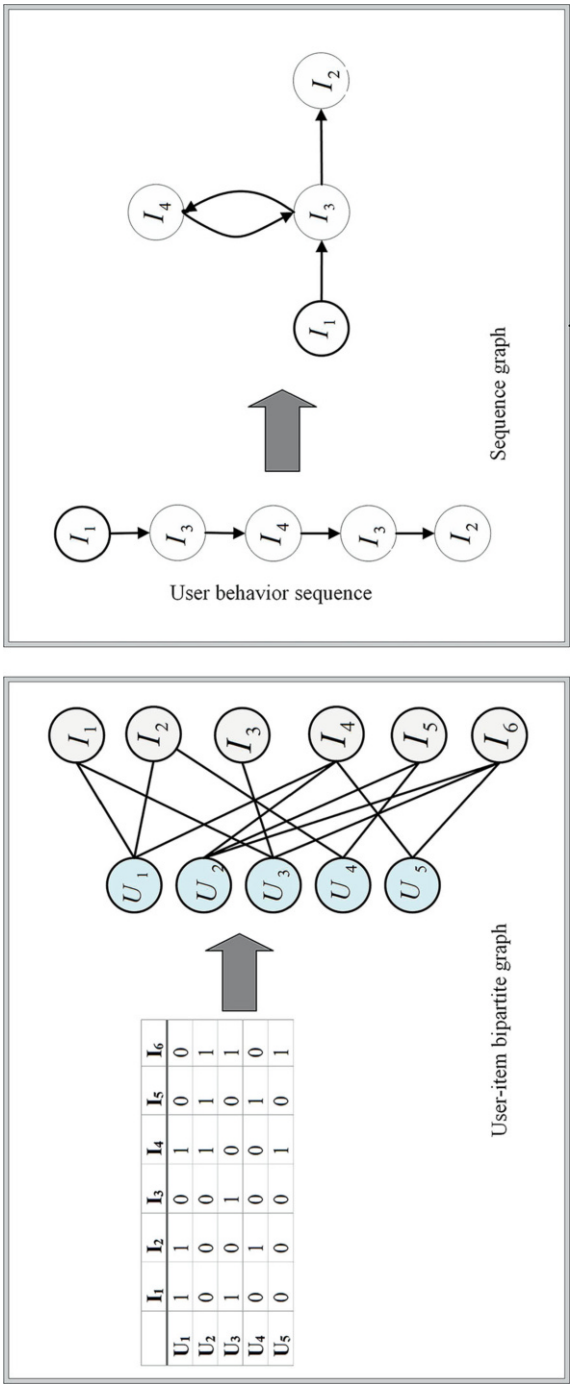

Fig. 5.12 An example of a bipartite and sequence graph in recommender systems

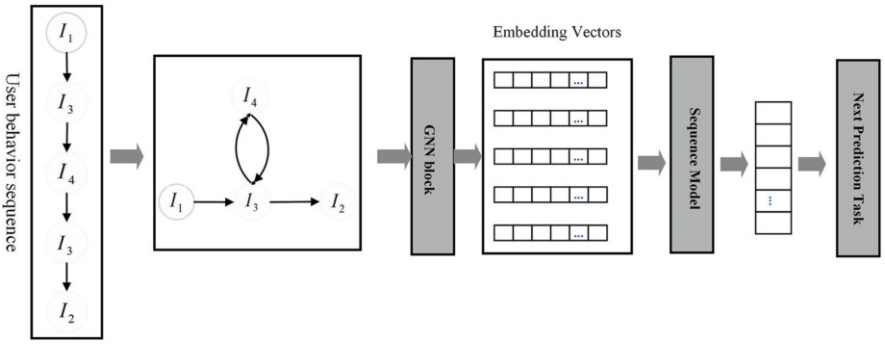

Fig. 5.13 The general framework of graph neural networks in SBRS

methods, all of which treat neighboring nodes equally [96]. The attention mechanism can also be used to distinguish the importance of neighbors [33, 54, 97]. All these methods adopt the permutation-invariant aggregation function during message transmission and ignore the sequence of items in the neighborhood, which may lead to information loss [50]. Several methods have been proposed to preserve the sequence of items in the graph construction [50].

In the subsequent subsections, first, different research which used graph neural networks in session-based recommender systems have been discussed. Then, the combination of GNNs and RNNs and, after that, the research that has used GCNs are reviewed.

5.4.2 Approaches Based on GNN

Session-based recommender systems that have used different types of graph neural networks face three main challenges: graph construction, information propagation, and sequential interests. Sequential data should be transformed into a sequence graph to build a graph, and it should be determined whether it is sufficient to create a subgraph for each sequence independently. Or is it better to add an edge between several consecutive items or only consider an edge between two consecutive items? To propagate information, it should be determined which propagation mechanism is more suitable for recording transition patterns. Is it necessary to recognize the order of related items? For the last problem, which is the user's sequential interests, item representations must be integrated into a sequence to obtain the user's temporal preference. Should one simply use attention-based pooling or a recurrent neural network structure to improve sequential temporal patterns?

Different answers to the above questions will lead to the proposal of a new approach, some of them will be discussed in the following.

Wu et al. have proposed a Session-based Recommendation with Graph Neural Networks, SR-GNN, that models session sequences as graph-structured data [36]. In SR-GNN, in the first step, the sequence of sessions is modeled as a directed graph with weighted edges, and each session sequence is considered a subgraph. In this graph, each node is an item, and each edge between two nodes represents the sequence of user clicks on two items. For example, if there is an edge from node v_i to node v_j, it means that the user clicked on v_j in a session after clicking on v_i. The weight of each edge is also calculated based on the occurrence of the edge divided by the outdegree of that edge's start node. Then, using the gated graph neural network, transitions and connections between the items are detected, and the embedding vectors of the items are generated.

Now, the information propagation between different nodes is performed based on the matrix of connections and edge weights, and the latent vectors of nodes are extracted and sent as input to the graph neural network. Then, two update and reset gates decide what information to keep and discard, respectively. After that, the candidate state is built based on the previous state, the current state, and the reset gate. After updating all the nodes in the session graph and reaching convergence, the final node vectors are obtained. Each session is presented using an attention network as a combination of global interest and the current interests of that session. Finally, the probability that each item will be the next click for each session is predicted. The general architecture of SR-GNN is shown in Fig. 5.14.

Pang et al. presented another model based on a heterogeneous graph neural network (HGNN) that attempts to recognize the simultaneous occurrence of items globally [31]. This work is done by creating the item-user interaction graph and the global co-occurrence graph of items. The global graph contains item transitions of sessions, user-item interactions, and the global co-occurrence items. To detect the transfer and change of items, first, a heterogeneous global graph including user and item nodes is constructed, and the user's previous interactions with items are used to create user-item edges. This leads to the detection of the long-term preferences of the user. Then, to generate connections between items, pairwise item transitions in the sessions are considered. To detect potential correlations, pairs of similar items are computed based on global co-occurrence information to construct item edges.

In HGNN, a graph augmented hybrid encoder consisting of a heterogeneous graph neural network and a personalized session encoder is proposed to create a session preference embedding to provide personalized session-based recommendations. The personalized session encoder combines current session item information and general user preferences to create a personalized session presentation, which is used to generate a more detailed and personalized list of recommendations. Since session-based recommender systems only provide recommendations based on the sequence of items that anonymous users have interacted with in a limited time frame, HGNN can be used in limited situations where the user ID and previous sessions are available and can be used [98].

In most methods based on graph neural networks, information is propagated between adjacent items, and the information on items that are not directly related is ignored. Multilayer graph neural networks were also used for information

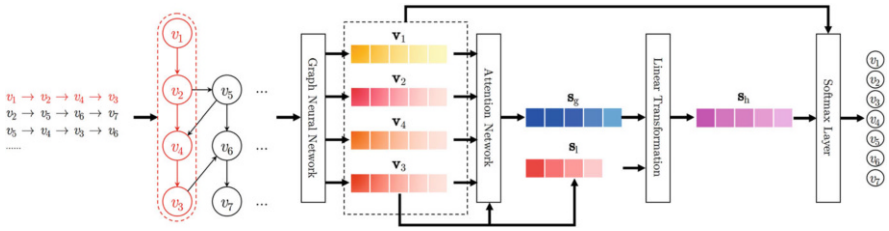

Fig. 5.14 The architecture of SR-GNN [36]

propagation between items without direct connections, but they also easily led to overfitting. To solve this challenge, Pan et al. have presented a Star Graph Neural Networks with Highway Networks (SGNN-HN) for session-based recommender system [38]. In SGNN-HN, a star graph neural network is used to model complex change patterns in the current session, which can solve the problem of long-term information propagation by adding a star node to check non-adjacent items. Then, to solve the overfitting problem of graph neural networks, highway networks are used to dynamically select the item embedding vectors before and after the star graph neural network, which can help to discover the complex transition relationship between items. Finally, the item embeddings generated by the star graph neural network are carefully integrated into the current session, and the proposed item is predicted based on the user's preferences.

It is worth mentioning that in SGNN-HN, a star graph is created for each session, and the set of graph nodes has two parts: the first part is all the unique items in a session, which are called satellite nodes, and the second part is the star node. Graph edges also include two types of connections for information propagation, which include satellite connections and star connections. Satellite connections are used to show adjacent relationships between items. For star connections, a bidirectional edge is added between the star node and each satellite node in the star graph, and on the other hand, to update the satellite nodes, edges are used to connect from the star node to the satellite node. Through the star node, information from non-adjacent nodes can be propagated in a two-hop way by considering the star node as an intermediate node. On the other hand, the directed edges from the satellite nodes to the star node are used to update the star node, which helps to create an accurate representation of the star node by considering all the nodes in the star graph. The general architecture of SGNN-HN is shown in Fig. 5.15.

Studying the methods' effectiveness in session-based recommender systems based on recurrent neural networks or attention-based mechanisms shows that complex item transitions decrease system efficiency. While graph neural networks adopt a mechanism that transforms snapshots of sessions into individual graphs at different timestamps to model static structural information without considering the temporal evolution of item transition relations, most session-based recommender system use cross-entropy with softmax to optimize model parameters, where all items (except the target item) are considered negative samples. During training, persistently decreasing negative item scores may cause overfitting of the model and loss of

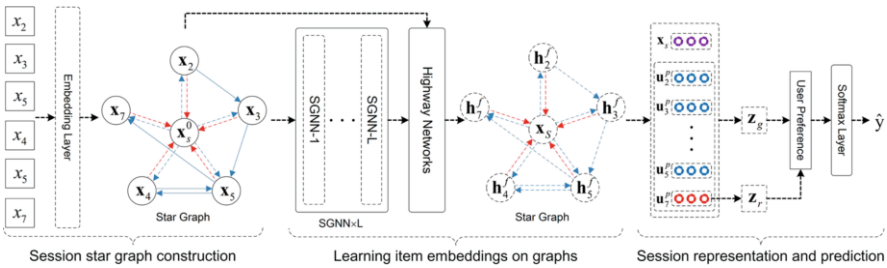

Fig. 5.15 The architecture of SGNN-HN [38]

generalization ability. Moreover, these approaches cannot provide a large enough gradient to reduce scores, limiting the convergence speed of the model.

To solve the abovementioned problems, dynamic graph learning for session-based recommender system (DGL-SR) is proposed by Pan et al. [32]. DGL-SR first converts the current session into a dynamic graph, and then, the structure layer is used to consider the structural information to learn the representation of session items at the different timestamps. At the same time, the structural information is analyzed through the graph attention network and the temporal evolution of the graph structures in different timestamps by temporal attention. To detect the temporal evolution of session graphs at different timestamps, DGL-SR produces a representation with the temporal features of each item using the temporal layer in the dynamic graph neural network. Then, the user's dynamic preferences are generated and used to generate predicted scores on all candidate items. Finally, a corrective margin softmax is developed to correct the gradient of negative items to avoid overfitting and achieve effective model optimization.

Qiu et al. proposed an improved graph neural network to learn the embeddings of each item in a session [33]. FGNN (full graph neural network) performs the learning process of embedding items by examining the inherent order of item transitions in a weight graph attention network. Therefore, it considers the pattern of item transitions by constructing a session graph and proposes a new model that jointly considers the sequence and hidden order in the session graph for a session-based recommender system. To use graph neural networks, a graph is constructed for each session, and the recommendation of the next item in the session is formulated as a graph-level classification problem. Specifically, it provides a weighted attention graph layer and a readout function to learn item embeddings and sessions to recommend the next item. The intrinsic sequence of the item transition pattern, which is critical for item-level feature representation, is achieved using a multiple-weighted graph attention layer network to compute the flow of information between items in a session. After generating the item representation, the readout function, which automatically learns to determine an appropriate order, is deployed to aggregate these features. Figure 5.16 shows the architecture of FGNN [33].

Many researchers attempted to provide methods that correctly identify the interests of anonymous users in short sessions. To this end, Li et al. proposed

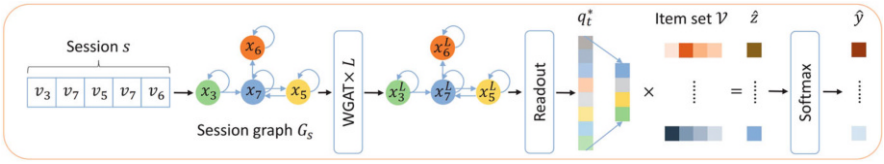

Fig. 5.16 The architecture of FGNN [33]

disentangled representation learning to create better representations for items in session-based recommender systems [34]. Disentangled Graph Neural Network (Disen-GNN) captures the purpose of ease session considering factor-level attention on each item. Disen-GNN consists of four main steps:

1. Embedding initialization: In this step, each session is converted into a directed graph, where each item in the session is encoded in embedding vectors with K parts. Each part represents the features of a factor. The similarity between adjacent items within a session is measured based on the features of each factor.
2. Disentangled item embedding learning: The factor-based similarity matrix is introduced, which estimates the similarity between adjacent items based on each item's embeddings. Then, the similarity matrix is integrated into the layers of the gated graph neural network to learn factor-based item embedding. A residual attention-based mechanism is also designed to preserve the distinctiveness of each item to avoid over-smoothing.
3. Session embedding learning: To detect the user's goal in a session, an attention-based network is used to calculate the user's attention to various factors of each item based on the factors of the last item, which provides the user's local goal. With assigned attention weights, a session embedding is created by weighted summing the factor embeddings of all items in the session.
4. Prediction: For each candidate item, the probability that the next item will be the user's choice is calculated by matching its embedding with the session embedding.

Considering the adjacent sessions and the correlation between them could be an influential factor for the development of session-based recommender systems. To this end, Pan et al. proposed a collaborative graph learning method (CGL) that utilized gated graph neural networks to learn item embeddings [42]. CGL consists of two main components: the main supervised and self-supervised modules. A gated graph neural network is used to present each item in each session. In the main supervised module, the user's recent and long-term interests are considered, and dynamic interest migration is detected. In this module, the model training is performed based on the supervised signals generated in sequential order, and the target-aware label confusion is designed to create an accurate label distribution. Label-aware confusion is used to generate soft labels that combine optimization with a one-hot encoding vector to avoid overfitting. Then, in the self-supervised module, supervision signals are extracted from the correlations between different sessions based on the general graph built to enrich the item representations. This general graph is

created based on all sessions, where each session is a node and its edges are defined based on similarity measure and max sampling. Based on this, the supervision signals are extracted from the correlation between sessions by self-supervised learning, and finally, the parameters of the model are optimized and updated based on the loss function of both components.

Due to the influence of friends' interests on each other's preferences, social networks have been utilized in many applications of session-based recommender system. Social network data is effective in better understanding users' interests and providing more accurate recommendations. In [40], an efficient framework for session-based social recommendation is proposed by Chen et al., in which, first, a heterogeneous graph neural network is used to learn user and item representations, which integrates the knowledge of social networks. Then, to generate predictions, only the user and item presentations related to the current session are sent to a non-social aware model. This framework has two advantages. First, the framework is flexible since it is compatible with existing non-social models and can easily incorporate more knowledge than social networks. Second, this framework can capture cross-session item transitions, whereas most existing methods can only capture intra-session item transitions.

5.4.3 Approaches Based on GNN and RNN

Due to adapting the RNN's functionalities to the sequential nature of data in session-based recommender systems, many researchers are interested in using these types of deep neural networks. Many session-based recommender systems usually model sequential signals using RNN's structures and transition relationships between items using GNNs to identify user interest. Of course, in real scenarios, there may be important and influential sequential signals in the behaviors of close users or multi-step transition relationships between different items. Therefore, the methods based on RNNs or GNNs can only get limited information to model the complex behavior patterns of users. Recurrent neural networks usually focus on the sequential relationships between items, whereas graph neural networks mainly focus on structural information. Therefore, these two models can be used together in session-based recommender systems to generate more acceptable results.

Chen et al. have proposed a collaborative co-attention network using the combination of recurrent neural network and graph neural network methods for session-based recommender systems [46]. In CCN-SR (Collaborative Co-attention Network for Session-based Recommendation), an embedding layer is considered for generating item embedding, whose outputs are the inputs of recurrent neural networks and graph neural networks. In the first step of CCN-SR, the user's behaviors in the current session, which includes the user's interaction items, are embedded and entered as input to the GRU to model the sequential relations between the user's interactions and behaviors. Since each step's hidden state contains sequential information between the user's previous behaviors up to this step and the user's current

goal, the hidden state of each step is modeled in the current session and collected by the recurrent neural network structure. Then, the transition relations between the items are modeled, and the detailed embeddings of the items in the current session are created with a graph neural network.

CCN-SR creates a directed graph for each session, whose nodes, items of each session, and edges indicate the sequence of user interaction with the items. An edge is generated between two nodes related to items in this graph when the user interacts with one item after another. Since some edges may appear multiple times in a session, these edges are given different weights to determine their importance. The weights are calculated based on the occurrence of the edge divided by the outdegree of the start node of the edge. Then, two adjacency matrices that specify the connections between nodes based on input and output edges are used in the graph neural network. Therefore, the encoder based on the graph neural network mainly recognizes the transition relations between adjacent items and models the structural information in the session. The output of the encoder based on the recurrent neural network contains the sequence information in the session. Combining these two types of information is helpful in providing a comprehensive representation for predicting recommendations.

In CCN-SR, two mechanisms are considered based on the co-attention mechanism: parallel co-attention and alternating co-attention. In the parallel mode, the structural information related to the graph neural network and the sequential information related to the GRU are taken as input, and it calculates the co-dependent representations at the same time in parallel. But in the alternating mode, it alternates between the initial outputs of the RNN-based session encoder and the GNN-based session encoder and the attentive representations of them. It should be mentioned that CCN-SR has high computational complexity and does not obtain acceptable results in sparse datasets. The general architecture of CCN-SR is shown in Fig. 5.17.

Session-based recommender systems that use graph neural networks can model data in a graph format and use content information and relationships between them to predict user behaviors. In addition, context-based recommender methods can integrate context data from several different and heterogeneous sources to obtain information about item features and relationships. To this end, Li et al. have proposed a session-based recommender system combined with two types of contextual information and work using gated graph neural networks, called context-aware and gated graph neural networks (CA-GGNN) [48]. Compared to the graph neural network, the gating graph neural network uses GRU and creates a message propagation model. The output of each layer of a typical GRU contains the current input information and the previous state information, which are specified by the input matrix and the state matrix, respectively. CA-GGNN first considers different types of input context information and time interval context information to dynamically create context matrices based on input information. These matrices include the input matrix and the time interval matrix. The input matrix represents the external environment information when the user makes the current decision, such as time and location. The time interval matrix presents the proportion of the time interval between the current decision and the next decision in the entire review period of

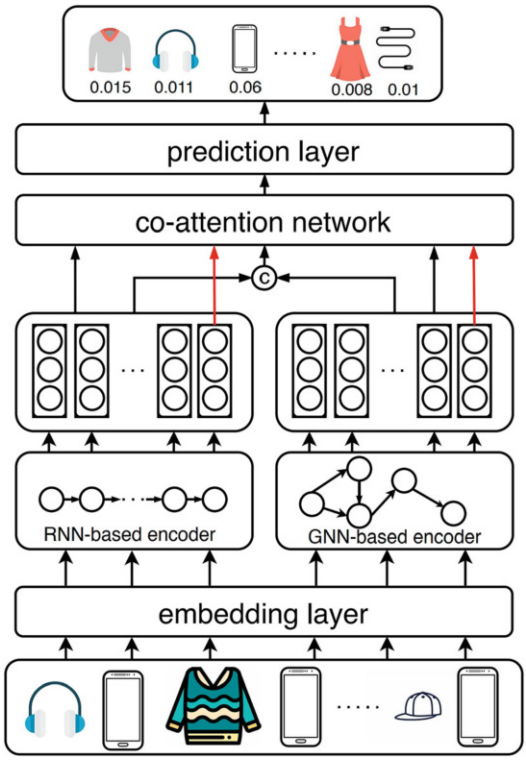

Fig. 5.17 The architecture of CCN-SR [46]

the session. In other words, the input matrix shows the scenario information of the external environment when the user participates in the session. The time interval matrix shows the proportion of time the user spends browsing each item in the entire session time. Then, the CA-GGNN replaces the constant input matrix and the state matrix with the input matrix and the interval matrix, respectively, and uses a context matrix to model the effect of the transformation of the input elements. Finally, it uses the time-based backpropagation method to train the model.

Figure 5.18 shows the general architecture of CA-GGNN [48], which includes three parts. Part "a" is related to data preprocessing and includes context data of the external environment, background data of time intervals, and the session graph based on the session sequence. In part "b," the gated graph neural network considers the graph structure information and the state information of each node at any time to create an accurate and reliable representation of each node. The process of this session-based recommender system depends not only on the sequence information of sessions but also on the sequence of sessions and related context information. In

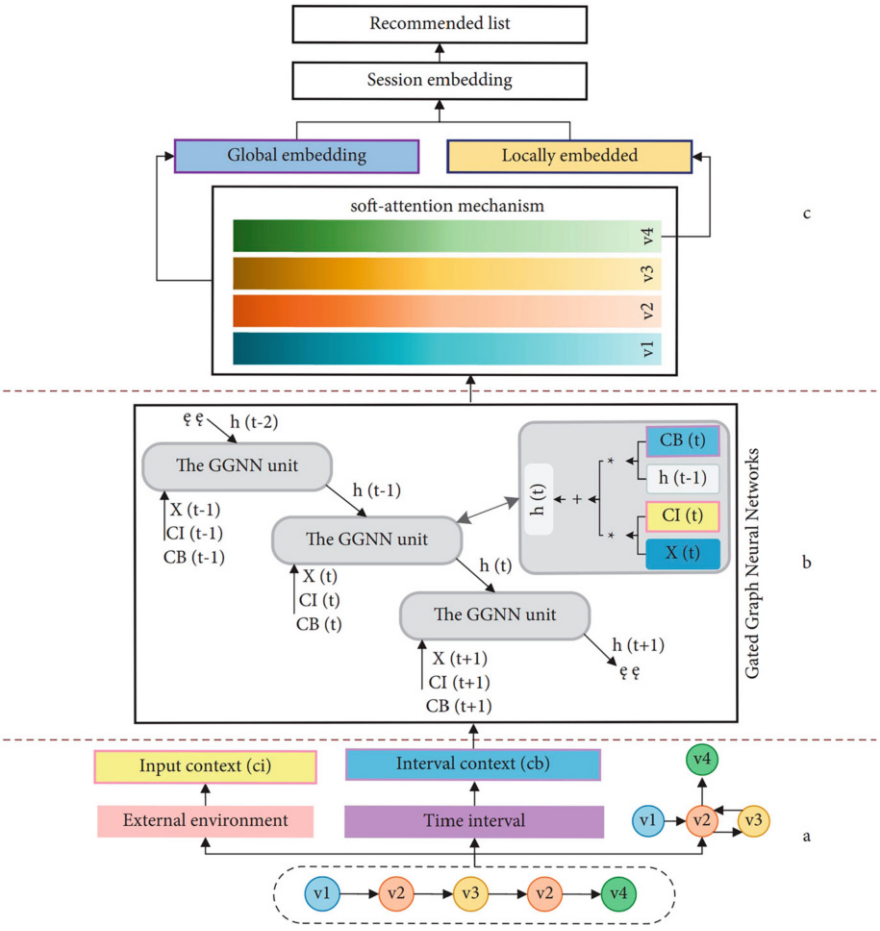

Fig. 5.18 The architecture of CA-GGNN [48]

part "c," CA-GGNN uses a soft attention-based mechanism to determine the priority between the user's long-term and short-term interests as the last item in the session sequence. A vector representation of the session sequence is obtained through a linear connection and used as a recommendation basis.

To overcome the problem of data sparsity and ignoring the effects of users' short-term and long-term interests on the accuracy of the recommendations, Hu et al. have proposed a session-based news recommender [47]. This method, GNewsRec (Graph Neural News Recommendation), first creates a heterogeneous user-news-topic graph to explicitly model users, news articles, and their topics based on the interactions that have already been performed between users and news articles. The topic information reflects the users' interests better and reduces the sparsity of user-item interactions. In GNewsRec, graph-based neural networks are used to encode relationships

between users, news articles, and topics. In this GNN model, user representations and news articles are learned by propagating their embeddings throughout the graph. The user's long-term interests are determined using the users' embedding based on the users' study history. In GNewsRec, the user's short-term interests are modeled using recent user studies and the LSTM model based on the attention mechanism.

GNewsRec consists of three main parts: a convolutional network for information extraction, a graph neural network for modeling news articles and long-term user interests, and LSTM based on an attention mechanism for modeling short-term user interests. The first part is composed of two parallel convolutional networks for extracting textual information, which takes the profile and title of the news as input and creates representations at the level of the profile and title of the news. Finally, both representations are placed next to each other. In the second part, a heterogeneous undirected graph of users, news articles, and topics is created to model users' long-term interests. In this method, in addition to the text of news articles, their sequence is also important. In the third part, LSTM based on the attention mechanism is used to detect the short-term interests of users. Figure 5.19 shows the architecture of GNewsRec [47].

Despite all advantages of using graph neural networks in session-based recommender system, some methods based on graph neural networks cannot express the sequential information of the session completely, e.g., the repeated nodes in a session and the starting node of the directed graph. On the other hand, noisy items inevitably exist in the sessions where the user chooses among diverse products with unintentional human behavior. This type of non-malicious noise is called natural noise, which has been less considered in session-based recommender systems. These problems limit GNN-based recommender methods and make them unable to achieve better results. To address these problems, Zhang et al. proposed a denoising graph neural network for session-based recommender system, called SEDGN (sequence-enhanced denoising graph neural network) [49]. SEDGN is a combination of GNN and GRU. It uses GRU to obtain sequential information to address limitations in session graph modeling. To reduce the effect of natural noise, two denoising modules have been developed separately in GNN and GRU, to produce two representation vectors. These vectors include the normal behavior information vector related to the sequence of the session and the graph vector of the session. Two denoising modules are designed to obtain normal user behavior information from sequential structure data and session graph structure, respectively, which reduce the effect of noise in the session. The item representations extracted from the sequentially structured data and graph structure are combined into a unified item representation that is used to predict the user's next click.

Another problem in session-based recommender systems is that the model does not deeply learn the potential characteristics of users and items when learning the deviation between user interests and recommended items. This leads to a certain degree of misunderstanding of the scope of users' interest preferences, leading to irrelevant or unexpected recommended content. In [52], an unexpected interest recommender system with a graph neural network (UIRS-GNN) is proposed by Xia et al. to address the current limitations of the models and uses a graph neural

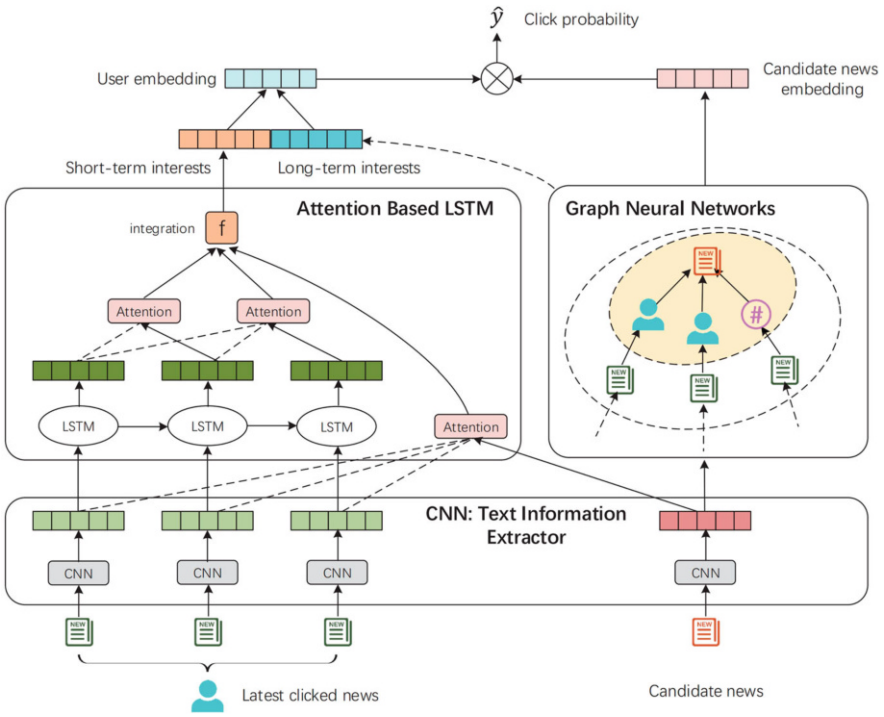

Fig. 5.19 The architecture of GNewsRec [47]

network to aggregate the features of neighboring nodes in the target node. UIRS-GNN can learn the user's preferences using an attention-based long short-term gated recurrent unit (A-LSGRU) network and model the user's general and local interests.

5.4.4 Approaches Based on GCN

Much data in the recommender system does not have a regular spatial structure. To model the complex relationship between users and items, a network structure that can process temporal and spatial information should be used. Graph convolutional networks can perform deep learning of spatiotemporal information on graph data.

Since the low-order approximation of GCN reflects short-term interest, in GACOforRec, the proposed method by Zhang et al., ConvLSTM, is used to ensure that the model can consider more conditions [63]. In addition to using temporal and spatial information learned by graph convolutional networks, this model uses LSTM capabilities to update and remember long-term preferences. Simultaneously, a new adaptive attention-based mechanism using convolutional graph networks is proposed to consider the effect of different propagation distances. To enhance the

hierarchical learning of the model from different priorities, a network structure called ON-LSTM, which focuses more on the hierarchy and sequence of the neuron, is introduced. This arrangement is necessary for a general understanding of the user's preferences.

In GACOforRec, the importance of user sessions is considered first. In real conditions, long-term historical records may not be critical to the user, and regular user activities are considered in one session. Therefore, considering the sequence of each user session and the connection between multiple sessions is an important goal of GACOforRec. GCN is used to model user sessions and learn sequences in the session and spatiality in the network to detect the short-term preferences of users. To avoid ignoring the long-term and persistent interests of the user, ConvLSTM is proposed, which is a type of recurrent neural network with long and short temporal effects while enabling the algorithm to focus on spatial domain information. This structure is used to connect two-part GCNs to account for "long" and "short" effects across the application scenario. ConvLSTM is used to combine connections and extract temporal information while paying attention to spatial feature extraction capabilities. Considering that different user behaviors may have different degrees of influence, a pair of new attention mechanisms are proposed that can obtain weights from different propagation distances in GCN.

The management and control of session-based recommender systems with many sessions and long time periods face three main challenges: First, the sessions are continuously growing. Memory cannot hold all sessions. Second, user interest may change significantly. A suitable model is required to model temporal information in past sessions. Third, the information in the new session should be modeled in time. To overcome these problems, Zhou et al. have proposed a temporal gated graph neural network that extracts auxiliary information from incoming sessions, called Temporal Augmented graph neural network for Session-based Recommendations (TASRec) [58]. TASRec dynamically models the user's long-term interest over a long period of time. Item-to-item interactions are presented on two levels: temporal graph and session graph. In the session graph, each node is an item related to the session, and each directed edge represents the adjacency of two items. For each day, a temporal graph is created that is undirected, and its nodes are the items of sessions before the desired day, and its edges are determined based on the adjacency of the items. The gated graph neural network is used for the session graph to determine the interactions of the items within the sessions, and based on that, the embedding of the items is obtained. In order to provide recommendations in the sessions of a certain day, the temporal modeling layer performs the learning of item interactions based on the graph that stores the records of the past sessions before the target day. An exponential denominator is used to change the scale of edge weight, and the effect of edge weight over time will decrease with the increase in the time difference between the previous day and the current day.

A multilayered graph convolutional network is also implemented to learn about high-order item interactions. Each layer of the GCN integrates all the first-order neighbor embeddings and the item itself. For the embedding of sessions, first, the embeddings within the session and the temporal embeddings of the item are

processed, and then the attention-based mechanism is used to compute the embedding of the session. Finally, based on the embedding of the session, the probability of selecting the candidate items is calculated.

Some data cannot be modeled using simple graphs. In these cases, a hypergraph can be used to model a more general data structure. A hypergraph consists of a set of vertices and a set of hyperedges, where a hyperedge can connect any number of vertices. This method is used to encode correlations of high-order data and has high ability and flexibility in detecting complex relationships in sessions. In this regard, the research [57] performs data modeling through hypergraphs in the context of GCN.

Xia et al. proposed a dual-channel hypergraph convolutional network (DHCN) [57]. DHCN models each session as a hyperedge in which all items are related to each other. Different edges connected through common items form a hypergraph that contains item-level high-order relations. By stacking multiple layers in the hypergraph channel, the strengths of hypergraph convolution can be used to produce efficient results. However, since each edge contains only a limited number of items, the problem of data sparsity may limit the benefits of hypergraph modeling. Therefore, the line graph channel is introduced, and self-supervised learning is integrated into the proposed model to enhance hypergraph modeling. A line graph is constructed on the basis of a hypergraph, where each hyperedge is a node and the edges are hyperedge connections that focus on session-level relations. After that, a dual-channel hypergraph convolutional network is developed, which describes two channels of information within and between sessions, while each of them knows little about the other. By maximizing the mutual information between session representations learned through two channels based on self-supervised learning, the two channels can acquire new information from each other to improve their performance in extracting item/session features. Figure 5.20 shows the architecture of DHCN [57].

Several session-based news recommender systems can extract expressive features (e.g., embedding) from articles and sessions, but they usually ignore the semantic-level structure information of articles. To this end, Sheu et al. proposed a novel context-aware graph embedding (CAGE) framework for session-based news recommendations that enriches news article embeddings using an auxiliary knowledge graph [65]. CAGE extracts textual-level features from news articles with convolutional neural networks while representing semantic-level entities with the help of a sub-knowledge graph. Semantic-level embedding extraction procedures include five steps, including entity extraction, triple extraction from an open knowledge graph, sub-knowledge graph construction, sub-knowledge graph embedding, and entity-linking embeddings. Then, CAGE refines the concatenated embeddings through multilayer graph convolutional networks. After that, session-level representations are learned by GRU. Finally, CAGE predicts the next click article for each session.

An issue that session-based recommender systems deal with is that they mainly rely on extracting sequential patterns in individual sessions. These methods are unexpressive enough to show more complex dependency relationships among

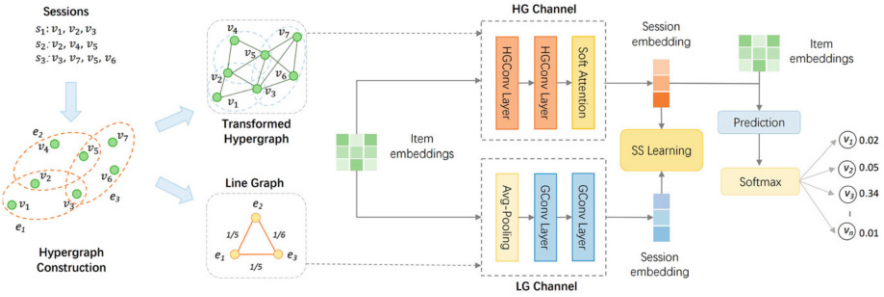

Fig. 5.20 The architecture of DHCN [57]

items. Additionally, due to the anonymity of session data, which prevents communication between different sessions, cross-session information is not taken into account. Some research, such as [60, 61], have proposed methods that are based on intersection sessions and have used graph neural networks with convolutional layers.

Ye et al. solved the above limitation, used sequential information between sessions, and proposed a cross-session aware temporal convolutional network (CA-TCN) [60]. For cross-sessions, CA-TCN constructs a cross-session item graph and a session-context graph to model the effect of cross-sessions on both items and sessions. The global cross-session item graph considers the effect of cross-sessions on the items by creating connections between items among all sessions, and the session-context graph by establishing connections between the current session and other sessions with interests. Similar user behaviors consider the complex interaction effect on sessions. Finally, items and sessions are connected by a hierarchical attention mechanism at the item level and the session level. Qiu et al. used the full graph neural network and modeled each session as a graph to learn the complex dependencies of items [61]. This method consists of two modules: (1) There is a weighted graph attention layer (WGAT) to encode information among nodes in the session graph for item embedding. (2) After obtaining the item embeddings, a readout function, which determines the dependencies of the items, is designed to aggregate the embeddings to produce a graph-level representation of the session embedding. In the last step, it outputs a list of ranked recommendations according to the comparison of session embedding with item embedding in the set of items.

5.5 SBRS Using Deep Reinforcement Learning

Before looking at the approaches of deep reinforcement learning models in session-based recommender systems, an overview of DRL and the reasons that made it an effective choice for SBRS are provided.

5.5.1 Why Deep Reinforcement Learning?

The reinforcement learning approach is more focused on goal-directed learning through interaction than other machine learning approaches. In reinforcement learning, the learner is not told what to do; rather, the agent must discover through trial and error and receiving rewards and punishments which actions bring the most rewards. The constituent elements of a reinforcement learning system are as follows:

- Agent: A program that is trained with the purpose of doing a specific task.
- Environment: The real or virtual world in which the agent performs actions.
- Action: A movement performed by an agent that causes a change in the state or condition of the environment.
- State: All information of the agent in its current environment.
- Observation: Observation is part of the situation that the agent can observe.
- Policy: Specifies what actions the agent will take given the current state. In the domain of deep learning, a neural network can be trained to make these decisions. During training, the agent attempts to improve its policy to make better decisions.
- Value function: Determines what is proper for the agent in the long-term execution. In other words, when the value function is applied to a given state, starting from that state, it yields the total expected reward that can be expected in the future.

The reinforcement learning cycle begins with the agent observing the environment (step 1) and receiving a state and a reward. Next, the agent uses this state and the reward to decide the following action (step 2). The agent then sends the action to the environment to control it in the desired way (step 3). Finally, the environment changes its state based on the previous state and the agent's action (step 4). Then, the cycle repeats. The reinforcement learning cycle is shown in Fig. 5.21.

In the reinforcement learning cycle at time t, the agent receives the state s_t from the environment and uses its policy model (π) to select a correct action a_t based on a control strategy. When the action is executed, the environment enters a new stage and provides the next state s_{t+1} as well as feedback in the form of a reward r_{t+1}. The agent uses the knowledge obtained during the state transition process in the form (s_t, a_t, s_{t+1}, r_{t+1}) to learn and improve π.

The reinforcement learning method suffers from the inefficient representation of features in problems with high dimensions or continuous agents. Therefore, learning time slows down, and some techniques must be developed to speed up the learning process. Considering that the most crucial feature of deep learning is the automatic generating of compact and low-dimensional representations of high-dimensional data using deep neural networks, a new domain called deep reinforcement learning was provided to help reduce the above problems. Deep reinforcement learning combines the advantages of deep learning and reinforcement learning to build efficient models in various scopes. In deep reinforcement learning, neural networks are used as agents to solve the reinforcement learning problem.

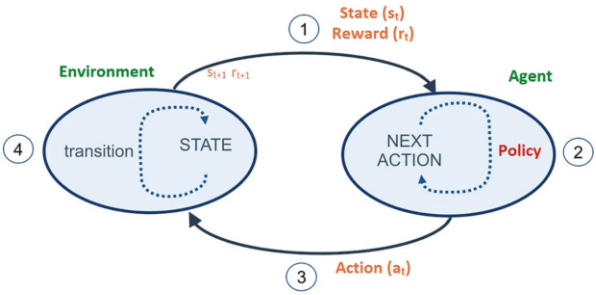

Fig. 5.21 The reinforcement learning cycle

Deep reinforcement learning can be divided into two categories: model-based and model-free methods. The main difference is how the agent learns from the environment. The goal of model-based methods is to estimate the transition function and reward function, while the goal of model-free methods is to estimate the value function or policy obtained from experience.

On the other hand, deep reinforcement learning approaches are divided into three streams: value-based, policy-based, and hybrid methods. In value-based methods, the agent updates the value function to learn a policy. Policy-based methods learn the policy directly, and hybrid methods combine value-based and policy-based methods, also called actor-critic methods. Actor-critic methods include two different networks, in which the actor network uses the policy-based and the critic network uses the value-based methods to evaluate the policy learned by the agent.

In terms of policymaking, deep reinforcement learning approaches can be divided into on-policy and off-policy methods. In off-policy, the behavioral policy π_b is used for exploration, while goal policy π is used for decision-making. In on-policy methods, the behavioral policy is the same as the goal policy.

One of the specific implementations of reinforcement learning approaches is the Q-Learning algorithm, which is a value-based approach utilizing the Q-Table concepts. Q-Table calculates the maximum expected reward for each action in each state. With this information, the model can choose the action with the maximum reward. The main idea behind Q-Learning is to use the Bellman optimization equation as an iterative update:

$$Q_{i+1}(s_t, a_t) = Q_i(s_t, a_t) + \alpha\left[R_i(s_t, a_t) + \gamma . \max Q_i\left(s'_t, a'_t\right) - Q_i(s_t, a_t)\right] \qquad (5.25)$$

In Eq. (5.25), γ is the discount rate, and α is the learning rate. Using the appropriate parameter γ makes rewards more controllable in the future. It is important to know that s'_t, a'_t comes from the behavioral policy π_b and s_t, a_t comes from the goal policy π. The optimal convergence of the Q-Function in many iterations will eventually be achieved ($Q_i \rightarrow Q^*, i \rightarrow \infty$).

In deep Q-Learning, a deep neural network is employed to approximate the Q-values' function, where the states are given as input and the Q-value of all possible actions is produced as output. The main difference between deep Q-Learning and

Q-Learning is the Q-Table implementation. Deep Q-Learning replaces the regular Q-Table with a neural network, and instead of mapping a state-action pair to a Q-value, a neural network maps input states to (action, Q-value) pairs. In fact, in deep Q-Learning, a function approximator, like a neural network with parameter θ, is trained to estimate Q-value so that $Q(s, a; \theta) \approx Q^*(s, a)$.

The difference between deep Q-Learning and Q-Learning is shown schematically in Fig. 5.22.

Some efforts have shown that reinforcement learning algorithms cope well with the problems of recommender systems based on sequential data because such problems can be naturally modeled as a Markov decision process to predict long-term user interests. Here, the recommender agent easily performs a sequence of ranking that usually learns the optimal policy from the recorded data with off-policy methods [11].

In these approaches, session-based recommender systems using reinforcement learning benefit from a recommender agent (RA) that interacts with the environment E (users) to achieve the maximum cumulative reward by sequentially selecting recommendation items during time steps. As discussed earlier, the modeling of this process includes a set of states, actions, and rewards. More formally, this set consists of five elements (S, A, P, R, γ) as follows:

- State space S: The state $s_t = \left(s_t^1, \ldots, s_t^n\right) \in S$ is defined as the browsing history of the user, that is, the n previous items that the user has checked before time t. The items in s_t are arranged in the order of occurrence.
- Action space A: Action $a_t = \left(a_t^1, \ldots, a_t^k\right) \in A$ is a list of items recommended to the user at time t based on the current state s_t, where k is the number of items that the recommender agent (RA) recommends to the user each time.
- Reward R: After the RA performs an action a_t in state s_t, i.e., recommends a list of items to the user, the user reviews these items and provides feedback, which can include skipping (not clicking), clicking, or ordering these items, and the agent receives the immediate reward $r(s_t, a_t)$ according to the user's feedback.
- Transition probability P: Transition probability $p(s_{t+1}|s_t, a_t)$ defines the probability of the state transition from s_t to s_{t+1} when RA performs an action a_t. If the user skips all the recommended items, then the next state is $s_{t+1} = s_t$, while if the user clicks/orders some items, then the next state is updated to s_{t+1}.
- Discount factor γ: The coefficient γ defines the discount factor when we measure the value of a future reward. In particular, when $\gamma = 0$, RA only considers the immediate reward. In other words, when $\gamma = 1$, all future rewards can be fully accounted for the reward of the current action.

Figure 5.23 shows the general framework of using deep reinforcement learning in session-based recommender systems.

At the end of this section, it should be mentioned that the nature of user interaction with a recommender system is sequential and the problem of recommending the best items to a user is not only a prediction problem but also a sequential decision problem. This can be solved by reinforcement learning

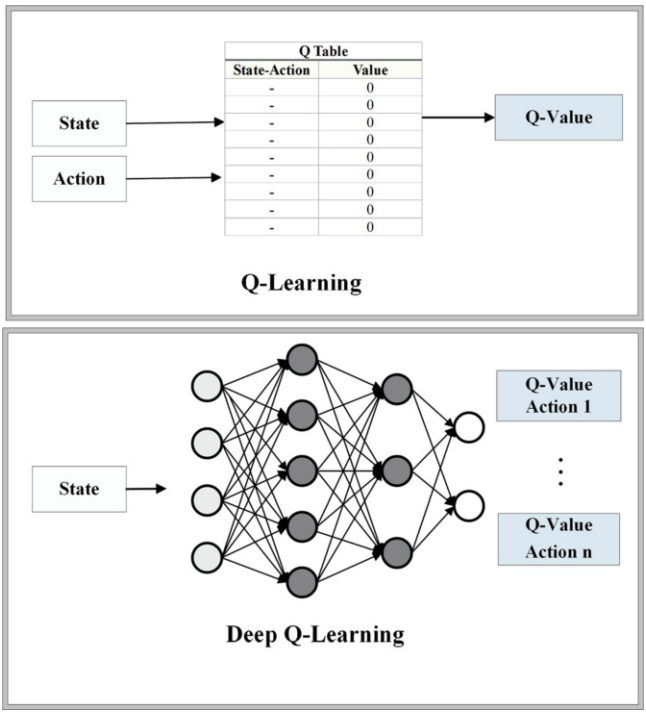

Fig. 5.22 The difference between deep Q-Learning and Q-Learning

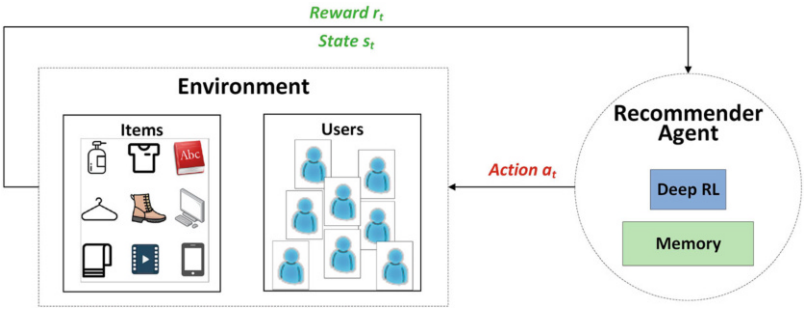

Fig. 5.23 The general framework of using deep reinforcement learning in SBRS

algorithms for three reasons. The first reason is that reinforcement learning can manage the dynamics of sequential user-system interaction by adjusting actions based on continuous feedback received from the environment. The second reason is that reinforcement learning can consider the long-term interaction of the user with the system. Finally, although it is beneficial to have user ratings, reinforcement learning naturally does not require user ratings and optimizes its policy by sequential

interaction with the environment. Because session-based recommender system work on sequential data and aim to consider users' sequential behavior, the reinforcement learning method could be suitable considering their characteristics.

The research reviewed in this section is mainly based on the use of deep reinforcement learning as the core. For this purpose, in Sect. 5.5.2, research based on the deep Q-Learning method is reviewed. Then, Sects. 5.5.3, 5.5.4, and 5.5.5 discuss the research using deep reinforcement learning combined with other deep neural networks, such as recurrent neural networks, convolutional neural networks, and generative adversarial networks.

5.5.2 Approaches Based on Deep Q-Learning

Q-Learning is a model-free, off-policy algorithm for learning the action values in a given state. This method is widely used in various domains and has also been used in session-based recommender system. To increase the capabilities of this method, deep Q-Learning is proposed, which is a type of Q-Learning that uses a deep neural network to represent the Q-Function instead of a simple table of values. Some research that use deep Q-Learning to provide session-based recommender systems are reviewed in this subsection.

Zhao et al. considered the sequential interactions between users and the recommender agent and used reinforcement learning to automatically learn optimal recommendation strategies [68]. This method is called LIst-wise Recommendation framework based on Deep reinforcement learning (LIRD). In fact, LIRD has presented a new session-based recommender system with the ability to continuously improve its strategies during interaction with users. Sequential interactions between users and a recommender system are modeled as a Markov decision process, and reinforcement learning is used to automatically learn optimal strategies through trial and error items' recommendations and receiving reinforcements for these items based on user feedback. An online user-agent interactive environment simulator is introduced in LIRD, which can pre-train and evaluate model parameters offline before applying the model online. Furthermore, the importance of listwise recommendations during interactions between users and the agent is confirmed, and a new approach is presented to apply them in a proposed framework for extensive list recommendations.

There are two main steps in the Actor framework of LIRD, which are the step of creating the parameters of the state-specific scoring function, which is performed through deep neural networks, and the step of creating the action, which is done based on the parameters of the scoring function of the previous step. The Critic framework is designed to use an estimator to learn an action-value function, which determines whether the action produced by the agent corresponds to the current state. This framework works based on deep Q-Learning. LIRD can be applied in scenarios with large and dynamic item spaces and can significantly reduce recalculations. Figure 5.24 shows the architecture of LIRD [68].

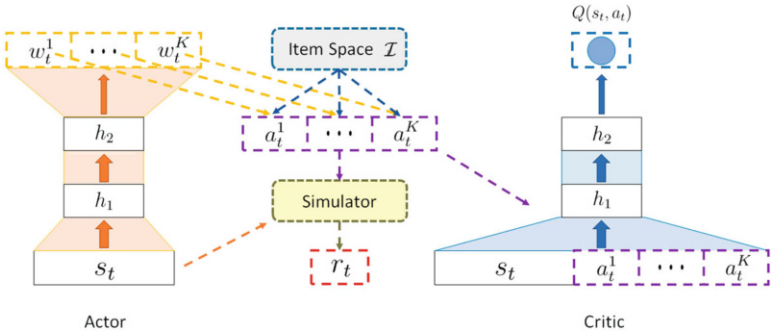

Fig. 5.24 The architecture of LIRD [68]

Zheng et al. proposed a deep reinforcement learning framework for online personalized news recommender systems, called Deep Reinforcement Learning Framework for News Recommendation (DRN) [69]. DRN uses deep Q-Learning to better model the dynamic and changing features of the news article and the user's interest so that it can simultaneously consider the current and future rewards. Deep Q-Learning architecture can easily increase scalability. A difference between DRN and other methods is that it considers user feedback as a combination of user clicks and the number of times the user returns to the news recommender system. Simultaneously, to provide more accurate recommendations and avoid irrelevant recommendations, it uses the Dueling Bandit Gradient Descent method.

In DRN, the environment consists of the collection of users and news articles, and the recommender algorithm plays the agent role. Feature representation for users is considered a state, and feature representation for news is considered as an action. When the user requests, a representation state (user features) and a set of action representations (candidate news features) are sent to the agent. The agent selects the best actions (recommends a list of news to the user) and receives the user's feedback. All feedback and recommendations are stored in the agent's memory. Every hour, the recommendation algorithm is updated based on the recommendation and feedback stored in the agent's memory.

5.5.3 Approaches Based on DRL and RNN

Many recommender systems treat the recommending process as static and provide recommendations following a fixed greedy strategy. However, these approaches may not be efficient enough due to the dynamic nature of user preferences. Furthermore, most existing recommender systems are designed to maximize the immediate (short-term) reward of recommendation while completely ignoring whether the recommended items lead to more efficient rewards in the future. To this end, Zhao et al. considered the recommendation as sequential interactions between users and

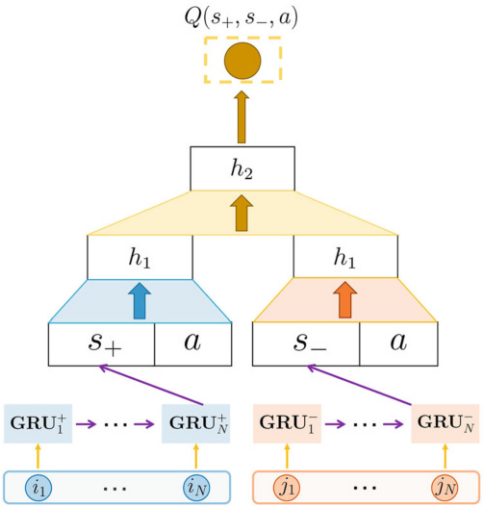

Fig. 5.25 The architecture of DEERS [71]

the recommender agent and used deep reinforcement learning to automatically learn optimal recommendation strategies [71]. The proposed deep reinforcement learning recommender system (DEERS) has two advantages. First, it can continuously update trial-and-error strategies during their interactions until the system converges to the optimal strategy to generate recommendations tailored to users' dynamic preferences. Second, the models in DEERS are trained by estimating the value of delayed rewards under current states and actions. Therefore, the system can quickly identify items with small immediate rewards that significantly affect future recommendation rewards.

Recommender systems based on reinforcement learning may not be flexible with the increasing number of items in the recommending process. This problem prevents their use in e-commerce recommender systems. For this purpose, a deep Q-Network (DQN) is used as a non-linear estimator to estimate the action-value function in DEERS. This model-free reinforcement learning method does not estimate the transition probability and does not store the Q-value table. This makes it flexible to support many items in recommender systems. It can also strengthen the system compared to traditional approaches that estimate the action-value function separately for each sequence.

In a recommender system, positive feedback indicates users' interests, and ignoring some of the recommended items by the user can help the system gain a better understanding of users' interests. Therefore, it is necessary to investigate such negative feedback, which is usually more than positive feedback. To this end, the DEERS framework considers negative feedback in addition to positive feedback and updates its strategy by receiving negative feedback.

Figure 5.25 shows the architecture of DEERS. This model concatenates the positive state and a recommended item as positive input (positive signals) and the negative state and a recommended item as negative input (negative signals). Then,

GRU is used to detect the preferences of the user, in which the update gate is used to create a new state and the reset gate is used to control the input from the previous state.

5.5.4 Approaches Based on DRL and CNN

Convolutional neural networks are widely used in various domains due to their ability to automatically extract temporal and spatial features individually or with other learning methods. A number of research reviewed in this section have used a combination of deep reinforcement learning and convolutional neural networks in session-based recommender systems.

To update the recommendation strategy according to the users' real-time feedback and create a page of items with an appropriate display, Zhao et al. developed the DeepPage approach to jointly generate a set of complementary items and the corresponding strategy to display them [75]. DeepPage is a novel page-wise recommendation framework based on deep reinforcement learning, which optimizes a page of items with an appropriate display based on real-time feedback from users. In this recommendation system, different factors are considered, such as the state, the user's current interests, actions, recommending a page of candidate items, and the reward that is the user's reaction, including clicking, skipping, or purchasing. The two fundamental challenges of applying deep reinforcement learning in such a problem are the large dynamic state space and the cost of computations for choosing the optimal action. To overcome these problems, the Actor-Critic framework is used, which is suitable for dynamic and large problems and reduces repetitive computations.

An encoder-decoder architecture is used in the Actor framework, in which GRU is used in the encoding part to generate the initial state, GRU. The GRU's input is the last clicked items before the current session, and its output is a vector of the user's initial interests. To learn the strategy of spatial representation of items on a page that leads to maximum reward, a convolutional neural network is used, the output of which is a dense vector with a low dimension and represents the items and user feedback on a given page. This vector is sent to another GRU to detect the real-time preferences of the user in the current session. In the decoder, a deconvolution neural network is used for reconstruction. The Critic framework is designed to use an approximator to learn the action-value function, which judges whether the proposed page generated by the Actor matches the current state. Figure 5.26 shows the architecture of DeepPage [75].

Gao et al. proposed DRCGR, using deep Q-Network and utilizing the CNN and GAN to help the agent to better understand high-dimensional data [76]. Two different convolution kernels have been used in DRCGR to capture positive feedback from users. Meanwhile, DRCGR uses a generative adversarial network to learn negative feedback to increase model robustness. DRCGR includes three main steps: The first step is to model the user's click behavior. At this stage, a matrix is formed

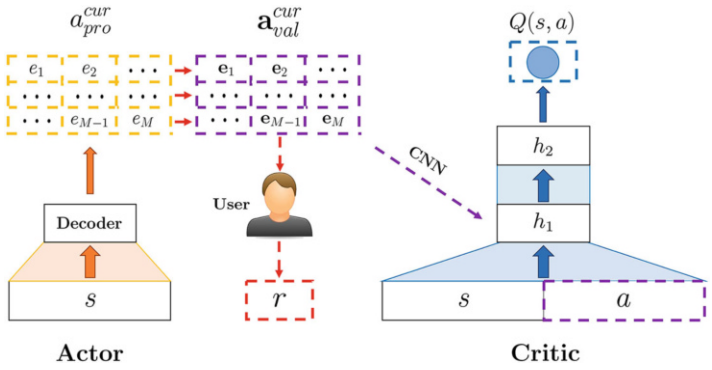

Fig. 5.26 The architecture of DeepPage [75]

on the basis of the embedding vectors of the items, and vertical and horizontal convolutional filters are applied to it, and the obtained results are placed next to each other. The second step creates more relevant negative feedback using a generative adversarial network. The third step integrates positive feedback and negative feedback in the DQN-based reinforcement learning model.

5.5.5 Approaches Based on DRL and GAN

One of the important challenges of reinforcement learning is the unstable convergence of models in the training process. In practice, the negative effect of reinforcement learning and recurrent neural networks is to increase the need for training data by the system, which is a challenge, particularly in session-based recommender system, which are inherently based on sparse data. One solution is to generate data through adversarial generative networks for the reinforcement learning method. Additionally, a random sampling method called the negative sampling method, which is used to show that the user is not interested in the items, cannot depict the user's preferences completely. So, the interests of the user may not be recognized correctly, and the items that are of interest to the user may be considered negative samples.

Zhao et al. addressed the problem mentioned above and proposed a collaborative filtering model based on deep adversarial generative network and deep reinforcement learning that uses the combination of Q-Learning and Actor-Critic models [77]. This method, Deep Generative Adversarial Networks-based Collaborative Filtering (DCFGAN), is an adaptive session-based recommender system presented in the domain of e-learning. DCFGAN uses the combination of deep adversarial generative network and deep reinforcement learning to take advantage of the user's immediate feedback and uses the deep deterministic policy gradient algorithm to return the gradient to increase the stability of the training process. Simultaneously,

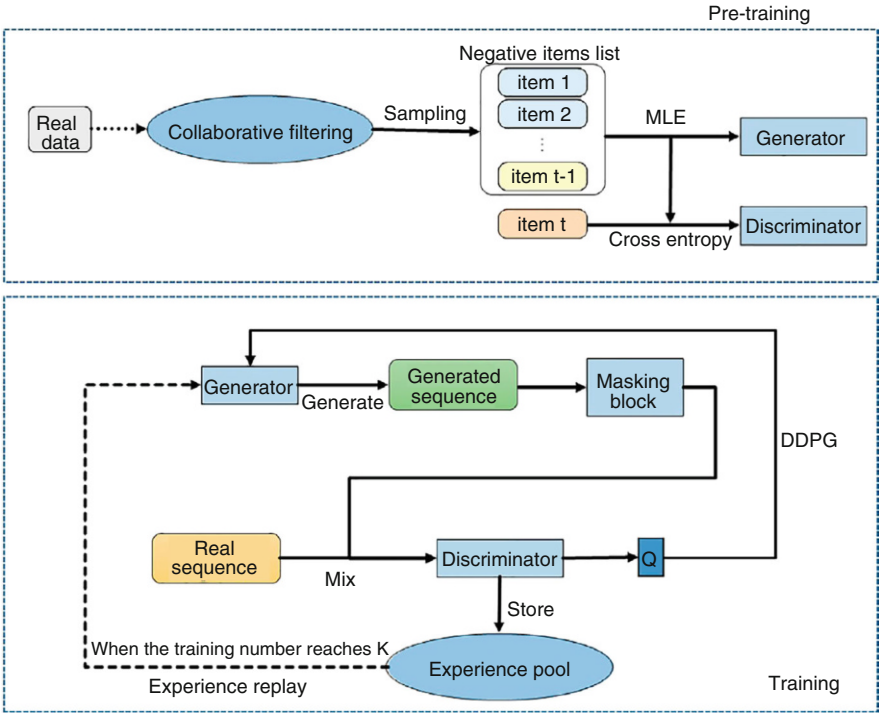

Fig. 5.27 The architecture of DCFGAN [77]

optimizing the value function using a deep deterministic policy gradient algorithm reduces the iterations required for convergence. According to the characteristics of session-based recommender system, DCFGAN uses pre-trained collaborative filtering in negative sampling items. It effectively improves the accuracy of negative sampling and is efficient for recommender systems. DCFGAN uses GRU to model sequential data in the deterministic part. Figure 5.27 shows the architecture of DCFGAN [77].

A model-based deep reinforcement learning framework for SBRS has been developed by Chen et al., where a GAN imitates user behavior dynamics and learns the reward function [78]. The authors also developed a novel Cascading DQN algorithm to obtain a recommendation policy that can handle a large number of candidate items. The cascading design of the action-value function allows to identify of the best subset of items from a large pool of candidates.

Gao et al. also proposed a deep reinforcement learning framework, DRCGR, that employs CNN and GAN models [76]. A CNN model is used to capture the sequential features for positive feedback, and a GAN model is adopted to learn optimal negative feedback representations. Then, positive/negative representations are fed into DQN simultaneously, which is claimed to generate a better action-value function.

5.6 Discussion

In this chapter, the methods based on hybrid/advanced deep neural network models in a session-based recommender system have been discussed and analyzed. Moreover, research has been reviewed using graph neural networks and deep reinforcement learning combined with other deep learning approaches.

Session-based recommender system approaches based on RNNs usually have a low speed and difficult training process for large volumes of data. The CNN-based methods have high memory consumption, and hidden representations are not interpretable and readable. Therefore, a large percentage of session-based recommender system use hybrid deep learning methods. A review of research in the session-based recommender system domain with hybrid deep neural network models shows that the most focus is on the following combinations: CNNs and RNNs, AEs and RNNs, different combinations based on GNNs, and DRLs plus other models such as CNN and RNN.

Generally, several research based on the combination of CNNs and LSTMs [22–26] recognize the features of the data using CNN and model the user behaviors based on LSTM. In various research, different types of CNNs are used, such as 3D-CNN, parallel CNN, etc., and each has its own characteristics.

Due to the high number of LSTM parameters, some research based on RNNs use GRU, which requires fewer parameters and limited computing resources. For this reason, many session-based recommender systems use hybrid deep neural network methods, using the combination of CNN and GRU [12–21].

Autoencoder also has been used with GRUs [27, 29] and LSTM [28] in session-based recommender systems. In these research, different types of autoencoders, such as stack autoencoders or denoising autoencoders, are employed to extract efficient representations for user interactions and feature transformations, and RNNs recognize sequential dependencies and long-term interests of the user.

In addition to hybrid deep neural network methods that consist of combining two or more types of single deep neural networks, there are two other types of advanced models, which consist of deep reinforcement learning and deep graph neural networks, which have been discussed in Sects. 5.4 and 5.5.

Many recommender systems' data have a graph structure, and graph neural networks are widely used in this field due to their high capability in graph data representation learning in different domains. On the other hand, due to their high flexibility, graph neural networks provide the capacity to easily model auxiliary data in addition to the main data. In session-based recommender systems, sequences of items can be modeled as graph-structured data to represent adjacency between items. Graph neural networks are widely used to identify the transition pattern from the sequential behaviors of users by converting them into the sequential graph. Research such as [31, 32, 33, 38] are based on graph neural networks, but other research have also combined graph neural networks with RNNs such as [34, 36, 45–48], or with CNNs (GCNs) such as [53–57].

Due to the over-smoothing problem, more studies focus on the appropriate augmentation of GNN layers (deeper GNN) to capture higher-order correlations on graphs and improve the performance of models [99, 100, 101]. Despite these advances, there is no standard solution for constructing very deep GNNs like CNNs, and related works suggest different strategies. Regarding future work, increasing the performance of deeper GNNs compared to current shallow GNNs is a fundamental challenge in the development of very deep GNNs, such as innovating works based on networks, while the computational and time complexity should also be acceptable.

Small-scale subgraph reconstruction from the original graph will be a suitable solution to overcome the scalability challenge. Sampling is a natural strategy that has been widely used for training large graphs. However, in sampling, relatively part of the information is lost. Few studies have focused on how to design an effective sampling strategy to balance the effectiveness and scalability. For example, GraphSAGE [93] randomly samples a fixed number of neighbors, and PinSage [102] uses a random walk strategy for sampling.

As mentioned in Sect. 5.4, GNN-based recommender models are mostly based on static graphs, while many dynamic factors exist in session-based recommender system. For example, user data are naturally collected dynamically in these systems. Moreover, modeling user dynamic preferences is one of the most important challenges in these recommendation scenarios. In addition, the platform may dynamically include new users, products, features, etc., which creates challenges for static graph neural networks. Recently, dynamic GNNs [103, 104] have attracted the attention of researchers who apply embedding propagation operations on dynamically constructed graphs.

Static graphs are stable, so they can be modeled practically, while dynamic graphs introduce changing structures. A serious future research challenge is how to design the GNN framework in response to dynamic graphs in practice. On the other hand, considering the characteristics of time evolution in session-based recommender system, the proposed model based on a dynamic GNN will be a promising research direction with broad applications in the real world.

Using a supervised approach on interactive data, relatively sparse results are obtained compared with the graph scale. Therefore, it is necessary to consider more supervised signals from the graph structure or recommendation task using self-supervised GNN. Various studies have tried strengthening GNN-based recommendations by designing auxiliary tasks from the graph structure with self-supervision [105, 106]. Data augmentation, such as node removal, can be used to generate sample pairs for contrastive training. We believe that using self-supervised tasks to learn meaningful and robust representations in session-based recommender system based on GNN is a suitable direction for future research.

Another learning method that widely has been utilized and discussed in this chapter is deep reinforcement learning. DRLs deal well with the problems of session-based recommender systems because such problems can be modeled as a Markov decision process to predict the user's long-term preferences. On the other hand, the nature of user interaction with a recommender system is sequential, which

is consistent with the interactive nature of reinforcement learning. By combining the advantages of deep learning and reinforcement learning, deep reinforcement learning tries to build effective platforms and has also been used in session-based recommender systems. There are several research such as [10, 68, 69] based on deep Q-Learning; [71–73, 77] based on the combination of DRLs and RNNs; and [75] and [76] based on the combination of DRLs and CNNs.

Most existing methods of SBRS use one agent. Multi-agent reinforcement learning (MARL) is a subfield of reinforcement learning that is capable of learning multiple policies and strategies. Although a single-agent reinforcement learning framework can only handle a single task, studies can be defined that consider the multi-task situation in SBRS and use multi-agent DRL (MADRL) or hierarchical DRL (HDRL). HDRL is proposed to handle complex tasks by dividing tasks into several small components and requires the agent to determine sub-policies. Different from HDRL, MADRL introduces several agents to handle subtasks. Hierarchical multi-agent RL (HMARL) combines HRL and MARL, where HDRL can be used to divide a complex task into several sub-tasks, such as users' long-term interests and short-term clicking behavior, and MADRL can also be considered several factors.

Sample inefficiency is a known challenge in model-free DRL methods used in SBRS. Model-free DRL requires a significant number of samples because there is no guarantee that the received mode will be useful. Typically, after a significant number of intervals, and after receiving a useful state and reward signal, the agent can begin to learn, which can be a severe challenge in the useful duration of a session. On the other hand, DRL model-based methods work more efficiently in this case, although they are more complicated because the agent must analyze the larger action and state space to learn the environmental model and the desired policy.

Unlike existing sequential models that use convolutional or recurrent networks, transformer is completely based on an attention mechanism called "self-attention," which is highly efficient and capable of uncovering syntactic and semantic patterns between words in a sentence [107]. Transformer uses self-attention to form a codec to calculate the contextual relationship. This network is used to weigh and aggregate the information of all items. For sequential recommendation, Kang and McAuley were the first to introduce a two-layer transformer decoder (i.e., transformer language model) called SASRec to capture user's sequential behaviors in 2018 [108]. In another research around the same time, Sun et al. proposed BERT4Rec, which employs deep bidirectional self-attention to model user behavior sequences [e]. After these significant and influential works in SBRS, many researchers focused on the transformer model and provided excellent performance solutions in SBRS [109–117].

Although a lot of work has been done using transformers in SBRS in the last few years, there is still much research potential:

- Incorporating rich context information into the transformer models, such as dwell time, action types, locations, devices, or any other context data.
- Transformer models that can handle very long sequences (e.g., clicks).

- Considering user's higher-order features and addressing the impact of item position information on the current session by utilizing the position attention layer in the transformer network.
- Since the transformers usually have a limited ability to identify local contextual information, a CNN model can be employed at the aggregating stage of the item features with long- and short-distance dependencies.
- Utilizing different time intervals in the behavior sequence of users that considers item relations and corresponding time intervals using the combination of GNN and transformers. This hybrid model can be embedded with time intervals to learn the complex interaction information among items and users.

Table 5.6 summarizes the existing works discussed in this chapter and addresses the application domain, deep learning model, type of input data, embedding technique, and loss function of each approach.

Table 5.6 A summary of the reviewed research

Ref.	Domain	Deep learning model	Input data	Embedding technique	Loss function
[22]	News	CNN + LSTM	Clicked news documents	PV-DBOW	BPR, TOP1
[23]	News	CNN + LSTM	Category, ID and keywords of items, sessions	Char-level embeddings	BPR, TOP1, cross-entropy
[24]	News	CNN + LSTM	Clicked news	Pre-trained from a large corpus or randomly initialized	Negative log-likelihood function
[25]	POI	CNN + LSTM	Items of session	D-dimensional vector	Cross-entropy
[26]	E-commerce	CNN + LSTM	Items of session + time series data	Label encoder	–
[15]	Job posting	CNN + GRU	Items of session	One-hot encoding + d-dimensional vector	Cross-entropy, BPR, noise contrastive estimation, L2 loss, hinge
[13]	News	CNN + GRU	News article + contextual data	CNN + pre-trained word embedding	Similarity loss function based on accuracy and novelty
[18]	E-commerce	CNN + GRU	Items of session	One-hot encoding	Cross-entropy
[19]	E-commerce	CNN + GRU	Items of session	One-hot encoding	Cross-entropy
[21]	E-commerce	CNN + Bi-GRU	Items of session	One-hot encoding + embedding lookup	Cross-entropy
[27]	Insurance	GRU + AE		One-hot encoding	Binary cross-entropy

(continued)

Table 5.6 (continued)

Ref.	Domain	Deep learning model	Input data	Embedding technique	Loss function
			All user actions across multiple sessions		
[29]	E-commerce	GRU + AE	Items of session	One-hot encoding + autoencoder	Loss function based on sum of mean square error
[34]	E-commerce	Gated graph neural networks (GGNN)	Items of session	D-dimensional vector + GGNN	Cross-entropy
[57]	E-commerce	Hypergraphs + GCN	Items of session	D-dimensional vector	Hybrid loss function based on cross-entropy
[48]	E-commerce	Gated GNN	Session sequences and related context information	Embedding vector representation of each item in the session graph (GGNN)	Cross-entropy
[63]	E-commerce	GCN + ConvLSTM	Items of session	D-dimensional node vector in directed session graph (GNN)	–
[75]	E-commerce	DRL + CNN	Category, embedding and, user's feedback of items of session	Pre-trained low-dimensional vector	DDPG
[69]	News	Deep Q-Learning	Interaction log	Continuous feature representation + one-hot encoding for news	–

5.7 Conclusion

Hybrid deep learning methods not only benefit from the advantages of single deep learning methods but also reduce the disadvantages of each method based on the capabilities that other models present. Due to the data complexity of session-based recommender systems, many approaches presented in this field are based on hybrid deep learning methods. Because of the sequential nature of the data, session-based recommender systems usually employ recurrent neural networks to model the sequence of events. Other deep learning methods can be combined with recurrent

neural networks to achieve more accurate feature extraction, achieve more optimal representations of inputs, and obtain better results.

In addition to hybrid deep neural network methods, two other types of advanced approaches are popular in session-based recommender systems: first, the approaches that utilize deep graph neural networks (GNNs) as the fundamental component, and second, the approaches that employ deep reinforcement learning (DRL) as the core module. Graph neural networks are a class of deep learning methods specifically developed to infer data described by graphs. In session-based recommender systems, it is possible to model the sequential behaviors and user-item interactions with a graph and learn the relations between users and items using a deep graph neural network. Moreover, graph neural networks could be combined with CNN and RNN models to provide more accurate and effective recommendations. Recommendation systems based on deep reinforcement learning benefit from a recommendation agent that interacts with the users to obtain the maximum cumulative reward by sequentially selecting recommendation items during time steps. Deep reinforcement learning also could be combined with CNN, GAN, and RNN models.

This chapter concluded with several discussions on the reviewed research and provided future directions and trends in session-based recommender systems using hybrid/advanced deep neural network models.

References

1. Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. "When and why are deep networks better than shallow ones?." In Proceedings of the AAAI conference on artificial intelligence, San Francisco, USA, February 4–9, 2017, vol. 31, no. 1. https://doi.org/10.1609/aaai.v31i1.10913
2. Cach N. Dang, María N. Moreno-García, and Fernando De la Prieta. "Hybrid deep learning models for sentiment analysis." Complexity 2021 (2021): 1-16. https://doi.org/10.1155/2021/9986920
3. Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. "Deep learning based recommender system: A survey and new perspectives." ACM computing surveys (CSUR) 52, no. 1 (2019): 1-38. https://doi.org/10.1145/3285029
4. Li Deng, and Dong Yu. "Deep learning: methods and applications." Foundations and trends® in signal processing 7, no. 3–4 (2014): 197-387. https://doi.org/10.1561/2000000039
5. Biswajit Jena, Sanjay Saxena, Gopal K. Nayak, Luca Saba, Neeraj Sharma, and Jasjit S. Suri. "Artificial intelligence-based hybrid deep learning models for image classification: The first narrative review." Computers in Biology and Medicine 137 (2021): 104803. https://doi.org/10.1016/j.compbiomed.2021.104803
6. Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. "Graph neural networks." In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 27-37. Springer, Singapore, 2022. https://doi.org/10.1007/978-981-16-6054-2_3
7. Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. "Graph neural networks in recommender systems: a survey." ACM Computing Surveys 55, no. 5 (2022): 1-37. https://doi.org/10.1145/3535101
8. Dai Hoang Tran, Quan Z. Sheng, Wei Emma Zhang, Abdulwahab Aljubairy, Munazza Zaib, Salma Abdalla Hamad, Nguyen H. Tran, and Nguyen Lu Dang Khoa. "Hetegraph: graph

learning in recommender systems via graph convolutional networks." *Neural computing and applications* (2021): 1-17. https://doi.org/10.1007/s00521-020-05667-z
9. Yao Ma, and Jiliang Tang. Deep learning on graphs. Cambridge University Press, 2021. https://doi.org/10.1017/9781108924184
10. Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. "Batch-constrained distributional reinforcement learning for session-based recommendation." *arXiv preprint arXiv:2012.08984* (2020). https://doi.org/10.48550/arXiv.2012.08984
11. Yuanguo Lin, Yong Liu, Fan Lin, Lixin Zou, Pengcheng Wu, Wenhua Zeng, Huanhuan Chen, and Chunyan Miao. "A survey on reinforcement learning for recommender systems." IEEE Transactions on Neural Networks and Learning Systems (2023). https://doi.org/10.1109/TNNLS.2023.3280161
12. Yupu Guo, Duolong Zhang, Yanxiang Ling, and Honghui Chen. "A joint neural network for session-aware recommendation." IEEE Access 8 (2020): 74205-74215. https://doi.org/10.1109/ACCESS.2020.2984287
13. Gabriel De Souza P. Moreira, Dietmar Jannach, and Adilson Marques Da Cunha. "Contextual hybrid session-based news recommendation with recurrent neural networks." IEEE Access 7 (2019): 169185-169203. https://doi.org/10.1109/ACCESS.2019.2954957
14. Lemei Zhang, Peng Liu, and Jon Atle Gulla. "Dynamic attention-integrated neural network for session-based news recommendation." Machine Learning 108 (2019): 1851-1875. https://doi.org/10.1007/s10994-018-05777-9
15. Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenburg, and Jure Leskovec. "Hierarchical temporal convolutional networks for dynamic recommender systems." In The world wide web conference, pp. 2236-2246. 2019. https://doi.org/10.1145/3308558.3313747
16. Xiao Gu, Haiping Zhao, and Ling Jian. "Sequence neural network for recommendation with multi-feature fusion." Expert Systems with Applications 210 (2022): 118459. https://doi.org/10.1016/j.eswa.2022.118459
17. Zhenyan Ji, Mengdan Wu, Yumin Feng, and José Enrique Armendáriz Íñigo. "Multi-channel Convolutional Neural Network Feature Extraction for Session Based Recommendation." Complexity 2021 (2021). https://doi.org/10.1155/2021/6661901
18. Ngo Xuan Bach, Dang Hoang Long, and Tu Minh Phuong. "Recurrent convolutional networks for session-based recommendations." Neurocomputing 411 (2020): 247-258. https://doi.org/10.1016/j.neucom.2020.06.077
19. Jinjin Zhang, Chenhui Ma, Xiaodong Mu, Peng Zhao, Chengliang Zhong, and A. Ruhan. "Recurrent convolutional neural network for session-based recommendation." Neurocomputing 437 (2021): 157-167. https://doi.org/10.1016/j.neucom.2021.01.041
20. Jingjing Wang, Lap-Kei Lee, and Nga-In Wu. "Dual-Channel Convolutional Recurrent Networks for Session-Based Recommendation." In Cyber Security, Privacy and Networking: Proceedings of ICSPN 2021, pp. 287-296. Singapore: Springer Nature Singapore, 2022. https://doi.org/10.1007/978-981-16-8664-1_25
21. Quan Li, Xinhua Xu, Jinjun Liu, and Guangmin Li. "Learning Sequential General Pattern and Dependency via Hybrid Neural Model for Session-Based Recommendation." IEEE Access 10 (2022): 89634-89644. https://doi.org/10.1109/ACCESS.2022.3201244
22. Keunchan Park, Jisoo Lee, and Jaeho Choi. "Deep neural networks for news recommendations." In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 2255-2258. 2017. https://doi.org/10.1145/3132847.3133154
23. Lemei Zhang, Peng Liu, and Jon Atle Gulla. "A deep joint network for session-based news recommendations with contextual augmentation." In Proceedings of the 29th on Hypertext and Social Media, pp. 201-209. 2018. https://doi.org/10.1145/3209542.3209557
24. Qiannan Zhu, Xiaofei Zhou, Zeliang Song, Jianlong Tan, and Li Guo. "Dan: Deep attention neural network for news recommendation." In Proceedings of the AAAI Conference on Artificial Intelligence, Hilton Hawaiian Village, Honolulu, Hawaii, USA, January 27 –

February 1, 2019, vol. 33, no. 01, pp. 5973-5980. https://doi.org/10.1609/aaai.v33i01.33015973
25. Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Jiajie Xu, Victor S. Sheng S. Sheng, Zhiming Cui, Xiaofang Zhou, and Hui Xiong. "Recurrent convolutional neural network for sequential recommendation." In The world wide web conference, pp. 3398-3404. 2019. https://doi.org/10.1145/3308558.3313408
26. Punam Bedi, Purnima Khurana, and Ravish Sharma. "Session Based Recommendations using CNN-LSTM with Fuzzy Time Series." In International Conference on Artificial Intelligence and Speech Technology, Delhi, India, November 12-13, 2021, pp. 432-446. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-95711-7_36
27. Simone Borg Bruun, Maria Maistro, and Christina Lioma. "Learning Recommendations from User Actions in the Item-poor Insurance Domain." In Proceedings of the 16th ACM Conference on Recommender Systems, Seattle, USA, September 18-23, 2022, pp. 113-123. https://doi.org/10.1145/3523227.3546775
28. Ivett Fuentes, Gonzalo Nápoles, Leticia Arco, and Koen Vanhoof. "Best Next Preference Prediction Based on LSTM and Multi-level Interactions." In Intelligent Systems and Applications: Proceedings of the 2021 Intelligent Systems Conference (IntelliSys), Amsterdam, Netherlands, September 1-2, 2022, Volume 1, pp. 682-699. Springer International Publishing. https://doi.org/10.1007/978-3-030-82193-7_46
29. Xin Chen, Alex Reibman, Sanjay Arora. "Sequential Recommendation Model for Next Purchase Prediction," In 4th International Conference on Advances in Artificial Intelligence Techniques (ArIT 2023), 2023, pp.141-158. https://doi.org/10.48550/arXiv.2207.06225
30. Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. "Graph Contextualized Self-Attention Network for Session-based Recommendation." In 28th International Joint Conference on Artificial Intelligence (IJCAI), Macao, August 10-16, vol. 19, pp. 3940-3946. 2019. https://doi.org/10.24963/ijcai.2019/547
31. Yitong Pang, Lingfei Wu, Qi Shen, Yiming Zhang, Zhihua Wei, Fangli Xu, Ethan Chang, Bo Long, and Jian Pei. "Heterogeneous global graph neural networks for personalized session-based recommendation." In Proceedings of the fifteenth ACM international conference on web search and data mining, pp. 775-783. 2022. https://doi.org/10.1145/3488560.3498505
32. Zhiqiang Pan, Wanyu Chen, and Honghui Chen. "Dynamic graph learning for session-based recommendation." Mathematics 9, no. 12 (2021): 1420. https://doi.org/10.3390/math9121420
33. Ruihong Qiu, Jingjing Li, Zi Huang, and Hongzhi Yin. "Rethinking the item order in session-based recommendation with graph neural networks." In Proceedings of the 28th ACM international conference on information and knowledge management, Beijing, China, November 3-7, 2019, pp. 579-588. 2019. https://doi.org/10.1145/3357384.3358010
34. Ansong Li, Zhiyong Cheng, Fan Liu, Zan Gao, Weili Guan, and Yuxin Peng. "Disentangled graph neural networks for session-based recommendation." IEEE Transactions on Knowledge and Data Engineering (2022). https://doi.org/10.1109/TKDE.2022.3208782
35. Feng Yu, Yanqiao Zhu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. "TAGNN: Target attentive graph neural networks for session-based recommendation." In Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, China, July 25-30, 2020, pp. 1921-1924. https://doi.org/10.1145/3397271.3401319
36. Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. "Session-based recommendation with graph neural networks." In Proceedings of the AAAI conference on artificial intelligence, Honolulu, Hawaii, USA, January 27 – February 1, 2019, vol. 33, no. 01, pp. 346-353. https://doi.org/10.1609/aaai.v33i01.3301346
37. Lifeng Yin, Pengyu Chen, and Guanghai Zheng. "Session-Enhanced Graph Neural Network Recommendation Model (SE-GNNRM)." Applied Sciences 12, no. 9 (2022): 4314. https://doi.org/10.3390/app12094314
38. Zhiqiang Pan, Fei Cai, Wanyu Chen, Honghui Chen, and Maarten De Rijke. "Star graph neural networks for session-based recommendation." In Proceedings of the 29th ACM international

conference on information & knowledge management, Ireland, October 19 - 23, 2020, pp. 1195-1204. https://doi.org/10.1145/3340531.3412014
39. Zhi-Hong Deng, Chang-Dong Wang, Ling Huang, Jian-Huang Lai, and S. Yu Philip. "G^ 3SR: Global Graph Guided Session-Based Recommendation." IEEE Transactions on Neural Networks and Learning Systems (2022). https://doi.org/10.1109/TNNLS.2022.3159592
40. Tianwen Chen, and Raymond Chi-Wing Wong. "An efficient and effective framework for session-based social recommendation." In Proceedings of the 14th ACM International Conference on Web Search and Data Mining, Israel, March 8 - 12, 2021, pp. 400-408. https://doi.org/10.1145/3437963.3441792
41. Wenjing Meng, Deqing Yang, and Yanghua Xiao. "Incorporating user micro-behaviors and item knowledge into multi-task learning for session-based recommendation." In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. China, July 25-30, 2020, pp. 1091–1100. https://doi.org/10.1145/3397271.3401098
42. Zhiqiang Pan, Fei Cai, Wanyu Chen, Chonghao Chen, and Honghui Chen. "Collaborative Graph Learning for Session-based Recommendation." ACM Transactions on Information Systems 40, 4 (2022), 1–26. https://doi.org/10.1145/3490479
43. Jianling Wang, Kaize Ding, Ziwei Zhu, and James Caverlee. "Session-based recommendation with hypergraph attention networks." In Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), April 29 - May 1, 2021. pp. 82-90. Society for Industrial and Applied Mathematics, 2021. https://doi.org/10.1137/1.9781611976700.10
44. Wen Wang, Wei Zhang, Shukai Liu, Qi Liu, Bo Zhang, Leyu Lin, and Hongyuan Zha. "Beyond clicks: Modeling multi-relational item graph for session-based target behavior prediction." In Proceedings of the web conference 2020, Taipei, Taiwan, April 20 - 24, 2020, pp. 3056-3062. https://doi.org/10.1145/3366423.3380077
45. Yupu Guo, Yanxiang Ling, and Honghui Chen. "A time-aware graph neural network for session-based recommendation." IEEE Access 8 (2020): 167371-167382. https://doi.org/10.1109/ACCESS.2020.3023685
46. Wanyu Chen, and Honghui Chen. "Collaborative co-attention network for session-based recommendation." Mathematics 9, no. 12 (2021): 1392. https://doi.org/10.3390/math9121392
47. Linmei Hu, Chen Li, Chuan Shi, Cheng Yang, and Chao Shao. "Graph neural news recommendation with long-term and short-term interest modeling." Information Processing & Management 57, no. 2 (2020): 102142. https://doi.org/10.1016/j.ipm.2019.102142
48. Dan Li, and Qian Gao. "Session Recommendation Model Based on Context-Aware and Gated Graph Neural Networks." Computational Intelligence and Neuroscience 2021 (2021). https://doi.org/10.1155/2021/7266960
49. Chunkai Zhang, Wenjing Zheng, Quan Liu, Junli Nie, and Hanyu Zhang. "SEDGN: Sequence enhanced denoising graph neural network for session-based recommendation." Expert Systems with Applications 203 (2022): 117391. https://doi.org/10.1016/j.eswa.2022.117391
50. Tianwen Chen, and Raymond Chi-Wing Wong. "Handling information loss of graph neural networks for session-based recommendation." In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, USA, July 6 - 10, 2020. pp. 1172-1180. https://doi.org/10.1145/3394486.3403170
51. Chen Chen, Jie Guo, and Bin Song. "Dual attention transfer in session-based recommendation with multi-dimensional integration." In Proceedings of the 44th International ACM SIGIR Conference on research and development in information retrieval, Online, July 11-15, 2021, pp. 869-878. https://doi.org/10.1145/3404835.3462866
52. Hongbin Xia, Kai Huang, and Yuan Liu. "Unexpected interest recommender system with graph neural network." Complex & Intelligent Systems (2022): 1-15. https://doi.org/10.1007/s40747-022-00849-9
53. Heng-Shiou Sheu, Zhixuan Chu, Daiqing Qi, and Sheng Li. "Knowledge-guided article embedding refinement for session-based news recommendation." IEEE Transactions on

Neural Networks and Learning Systems 33, no. 12 (2021): 7921-7927. https://doi.org/10.1109/TNNLS.2021.3084958
54. Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. "Global context enhanced graph neural networks for session-based recommendation." In Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, China, July 25-30, 2020, pp. 169-178. https://doi.org/10.1145/3397271.3401142
55. Tajuddeen Rabiu Gwadabe, and Ying Liu. "Improving graph neural network for session-based recommendation system via non-sequential interactions." Neurocomputing 468 (2022): 111-122. https://doi.org/10.1016/j.neucom.2021.10.034
56. Xiangde Zhang, Yuan Zhou, Jianping Wang, and Xiaojun Lu. "Personal interest attention graph neural networks for session-based recommendation." Entropy 23, no. 11 (2021): 1500. https://doi.org/10.3390/e23111500
57. Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. "Self-supervised hypergraph convolutional networks for session-based recommendation." In Proceedings of the AAAI conference on artificial intelligence, February 2-9, 2021, vol. 35, no. 5, pp. 4503-4511. https://doi.org/10.1609/aaai.v35i5.16578
58. Huachi Zhou, Qiaoyu Tan, Xiao Huang, Kaixiong Zhou, and Xiaoling Wang. "Temporal augmented graph neural networks for session-based recommendations." In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Canada, July 11-15, 2021, pp. 1798-1802. https://doi.org/10.1145/3404835.3463112
59. Xin Xia, Hongzhi Yin, Junliang Yu, Yingxia Shao, and Lizhen Cui. "Self-supervised graph co-training for session-based recommendation." In Proceedings of the 30th ACM International conference on information & knowledge management, Queensland, Australia, November 1-5, 2021, pp. 2180-2190. https://doi.org/10.1145/3459637.3482388
60. Rui Ye, Qing Zhang, and Hengliang Luo. "Cross-Session Aware Temporal Convolutional Network for Session-based Recommendation." In 2020 International Conference on Data Mining Workshops (ICDMW), Sorrento, Italy, November 17-20, pp. 220-226. https://doi.org/10.1109/ICDMW51313.2020.00039
61. Ruihong Qiu, Zi Huang, Jingjing Li, and Hongzhi Yin. "Exploiting cross-session information for session-based recommendation with graph neural networks." ACM Transactions on Information Systems (TOIS) 38, no. 3 (2020): 1-23. https://doi.org/10.1145/3382764
62. Gu Tang, Xiaofei Zhu, Jiafeng Guo, and Stefan Dietze. "Time enhanced graph neural networks for session-based recommendation." Knowledge-Based Systems 251 (2022): 109204. https://doi.org/10.1016/j.knosys.2022.109204
63. Mingge Zhang, and Zhenyu Yang. "GACOforRec: Session-based graph convolutional neural networks recommendation model." IEEE Access 7 (2019): 114077-114085. https://doi.org/10.1109/ACCESS.2019.2936461
64. Ruihong Qiu, Hongzhi Yin, Zi Huang, and Tong Chen. "Gag: Global attributed graph neural network for streaming session-based recommendation". In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 669–678. https://doi.org/10.1145/3397271.3401109
65. Heng-Shiou Sheu, and Sheng Li. "Context-aware graph embedding for session-based news recommendation". In Fourteenth ACM conference on recommender systems. Brazil, September 22-26, 2020. pp. 657–662. https://doi.org/10.1145/3383313.3418477
66. Yujia Zheng, Siyi Liu, Zekun Li, and Shu Wu. "Dgtn: Dual-channel graph transition network for session-based recommendation." In 2020 International Conference on Data Mining Workshops (ICDMW), pp. 236-242. IEEE, 2020. https://doi.org/10.1109/ICDMW51313.2020.00041
67. Diddigi Raghu Ram Bharadwaj, Lakshya Kumar, Saif Jawaid, and Sreekanth Vempati. "Fine-Grained Session Recommendations in E-commerce using Deep Reinforcement Learning." arXiv preprint arXiv:2210.15451 (2022). https://doi.org/10.48550/arXiv.2210.15451

68. Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. "Deep Reinforcement Learning for List-wise Recommendations." In 1st Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD 2019), USA, August 5, 2019. https://doi.org/10.48550/arXiv.1801.00209
69. Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. "DRN: A deep reinforcement learning framework for news recommendation." In Proceedings of the 2018 world wide web conference, Lyon, France, April 23 - 27, 2018, pp. 167-176. https://doi.org/10.1145/3178876.3185994
70. Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. "Self-supervised reinforcement learning for recommender systems." In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, China, July 25-30, 2020, pp. 931-940. https://doi.org/10.1145/3397271.3401147
71. Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. "Recommendations with negative feedback via pairwise deep reinforcement learning." In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, United Kingdom, August 19-23, 2018, pp. 1040-1048. https://doi.org/10.1145/3219819.3219886
72. Liwei Huang, Mingsheng Fu, Fan Li, Hong Qu, Yangjun Liu, and Wenyu Chen. "A deep reinforcement learning based long-term recommender system." Knowledge-Based Systems 213 (2021): 106706 https://doi.org/10.1016/j.knosys.2020.106706
73. Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. "Reinforcement learning to optimize long-term user engagement in recommender systems." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage USA, August 4-8, 2019, pp. 2810-2818. https://doi.org/10.1145/3292500.3330668
74. Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Jiliang Tang, and Hui Liu. "Dear: Deep reinforcement learning for online advertising impression in recommender systems." In Proceedings of the AAAI conference on artificial intelligence, February 2–9, 2021, vol. 35, no. 1, pp. 750-758. https://doi.org/10.1609/aaai.v35i1.16156
75. Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. "Deep reinforcement learning for page-wise recommendations." In Proceedings of the 12th ACM Conference on Recommender Systems, Vancouver, Canada, October 2, 2018 pp. 95-103. https://doi.org/10.1145/3240323.3240374
76. Rong Gao, Haifeng Xia, Jing Li, Donghua Liu, Shuai Chen, and Gang Chun. "DRCGR: Deep reinforcement learning framework incorporating CNN and GAN-based for interactive recommendation." In 2019 IEEE International Conference on Data Mining (ICDM), Beijing, China, November 8-11, 2019, pp. 1048-1053. https://doi.org/10.1109/ICDM.2019.00122
77. Jianli Zhao, Hao Li, Lijun Qu, Qinzhi Zhang, Qiuxia Sun, Huan Huo, and Maoguo Gong. "DCFGAN: An adversarial deep reinforcement learning framework with improved negative sampling for session-based recommender systems." Information Sciences 596 (2022): 222-235. https://doi.org/10.1016/j.ins.2022.02.045
78. Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. "Generative adversarial user model for reinforcement learning based recommendation system." In International Conference on Machine Learning, pp. 1052-1061. PMLR, 2019.
79. Xueying Bai, Jian Guan, and Hongning Wang. "A model-based reinforcement learning with adversarial training for online recommendation." Advances in Neural Information Processing Systems 32 (2019).
80. Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. "Sequence-aware recommender systems." ACM Computing Surveys (CSUR) 51, no. 4 (2018): 1-36. https://doi.org/10.1145/3190616
81. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. "Session-based recommendations with recurrent neural networks." In Proceedings International

Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016. https://doi.org/10.48550/arXiv.1511.06939
82. James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta et al. "The YouTube video recommendation system." In Proceedings of the fourth ACM conference on Recommender systems, Barcelona, Spain, September 26-30, 2010, pp. 293-296. https://doi.org/10.1145/1864708.1864770
83. Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. "Neural attentive session-based recommendation." In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, Singapore, November 6-10, 2017, pp. 1419-1428. https://doi.org/10.1145/3132847.3132926
84. Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. "STAMP: short-term attention/memory priority model for session-based recommendation." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 1831-1839. 2018. https://doi.org/10.1145/3219819.3219950
85. Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian personalized ranking from implicit feedback." In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal Quebec, Canada, June 18 - 21, 2009, pp. 452-461.
86. Mengqi Zhang, Shu Wu, Meng Gao, Xin Jiang, Ke Xu, and Liang Wang. "Personalized graph neural networks with attention mechanism for session-aware recommendation." IEEE Transactions on Knowledge and Data Engineering 34, no. 8 (2020): 3946-3957. https://doi.org/10.1109/TKDE.2020.3031329
87. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." nature 518, no. 7540 (2015): 529-533. https://doi.org/10.1038/nature14236
88. Hasselt Van, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In Proceedings of the AAAI conference on artificial intelligence, vol. 30, no. 1. 2016. https://doi.org/10.1609/aaai.v30i1.10295
89. Gabriel de Souza Pereira Moreira. "CHAMELEON: a deep learning meta-architecture for news recommender systems." In Proceedings of the 12th ACM Conference on Recommender Systems, Vancouver, Canada, October 2, 2018, pp. 578-583. https://doi.org/10.1145/3240323.3240331
90. Gabriel de Souza Pereira Moreira, Felipe Ferreira, and Adilson Marques da Cunha. "News session-based recommendations using deep neural networks." In Proceedings of the 3rd workshop on deep learning for recommender systems, Vancouver, Canada, October 6, 2018, pp. 15-23. https://doi.org/10.1145/3270323.3270328
91. Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32, no. 1 (2020): 4-24. https://doi.org/10.1109/TNNLS.2020.2978386
92. Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. "Graph neural networks: A review of methods and applications." AI open 1 (2020): 57-81. https://doi.org/10.1016/j.aiopen.2021.01.001
93. Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in neural information processing systems 30 (2017).
94. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018.
95. Thomas N. Kipf, and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In International Conference on Learning Representations ICLR 2017, Toulon, France, April 24-26, 2017.
96. Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. "Gated Graph Sequence Neural Networks." In Proceedings of ICLR'16. San Juan, Puerto Rico, May 2-4, 2016.

97. Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. "Sequential recommendation with graph neural networks." In Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval, Canada, July 11-15, 2021, pp. 378-387. https://doi.org/10.1145/3404835.3462968
98. Heeyoon Yang, Gahyung Kim, and Jee-Hyoung Lee. "Logit Averaging: Capturing Global Relation for Session-Based Recommendation." Applied Sciences 12, no. 9 (2022): 4256. https://doi.org/10.3390/app12094256
99. Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. "Towards deeper graph neural networks with differentiable group normalization." Advances in neural information processing systems 33 (2020): 4917-4928.
100. Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view." In Proceedings of the AAAI conference on artificial intelligence, New York, USA, February 7–12, 2020, vol. 34, no. 04, pp. 3438-3445. https://doi.org/10.1609/aaai.v34i04.5747
101. Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification." In International Conference on Learning Representations. Addis Ababa, Ethiopia, April 26-30, 2020, pp. 1–18.
102. Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. "Graph convolutional neural networks for web-scale recommender systems." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 974-983. 2018. https://doi.org/10.1145/3219819.3219890
103. Maosen Li, Siheng Chen, Yangheng Zhao, Ya Zhang, Yanfeng Wang, and Qi Tian. "Dynamic multiscale graph neural networks for 3d skeleton based human motion prediction." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Seattle, USA, June 13-19, 2020, pp. 214-223.
104. Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. "Streaming graph neural networks." In Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, pp. 719-728. 2020. https://doi.org/10.1145/3397271.3401092
105. Junliang Yu, Hongzhi Yin, Jundong Li, Qinyong Wang, Nguyen Quoc Viet Hung, and Xiangliang Zhang. "Self-supervised multi-channel hypergraph convolutional network for social recommendation." In Proceedings of the web conference 2021, Ljubljana, Slovenia, April 19-23, 2021, pp. 413-424. https://doi.org/10.1145/3442381.3449844
106. Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. "Self-supervised graph learning for recommendation." In Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval, pp. 726-735. 2021. https://doi.org/10.1145/3404835.3462862
107. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).
108. Wang-Cheng Kang, and Julian McAuley. "Self-attentive sequential recommendation." In 2018 IEEE international conference on data mining (ICDM), pp. 197-206. IEEE, 2018. https://doi.org/10.1109/ICDM.2018.00035
109. Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer." In Proceedings of the 28th ACM international conference on information and knowledge management, China, November 3-7, 2019, pp. 1441-1450. https://doi.org/10.1145/3357384.3357895
110. Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. "Transformers4rec: Bridging the gap between nlp and sequential/session-based recommendation." In Proceedings of the 15th ACM Conference on Recommender Systems, Netherlands, September 27-October 1, 2021, pp. 143-153. https://doi.org/10.1145/3460231.3474255

111. Chen Chen, Bin Song, Jie Guo, and Tong Zhang. "Multi-dimensional shared representation learning with graph fusion network for Session-based Recommendation." Information Fusion 92 (2023): 205-215. https://doi.org/10.1016/j.inffus.2022.11.021
112. Jingjing Wang, Haoran Xie, Fu Lee Wang, and Lap-Kei Lee. "A transformer–convolution model for enhanced session-based recommendation." Neurocomputing 531 (2023): 21-33. https://doi.org/10.1016/j.neucom.2023.01.083
113. Huanwen Wang, Yawen Zeng, Jianguo Chen, Ning Han, and Hao Chen. "Interval-enhanced Graph Transformer solution for session-based recommendation." Expert Systems with Applications 213 (2023): 118970. https://doi.org/10.1016/j.eswa.2022.118970
114. Yichao Lu, Zhaolin Gao, Zhaoyue Cheng, Jianing Sun, Bradley Brown, Guangwei Yu, Anson Wong, Felipe Pérez, and Maksims Volkovs. "Session-based Recommendation with Transformers." In Proceedings of the Recommender Systems Challenge 2022, Seattle, USA, September 18-23, 2022, pp. 29-33. https://doi.org/10.1145/3556702.3556844
115. Walid Shalaby, Sejoon Oh, Amir Afsharinejad, Srijan Kumar, and Xiquan Cui. "M2TRec: Metadata-aware Multi-task Transformer for Large-scale and Cold-start free Session-based Recommendations." In Proceedings of the 16th ACM Conference on Recommender Systems, Seattle, USA, September 18-23, 2022, pp. 573-578. https://doi.org/10.1145/3523227.3551477
116. Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. "SSE-PT: Sequential recommendation via personalized transformer." In Proceedings of the 14th ACM Conference on Recommender Systems, Brazil, September 22-26, 2020, pp. 328-337. https://doi.org/10.1145/3383313.3412258
117. Gabriel de Souza Pereira Moreira, Sara Rabhi, Ronay Ak, Md Yasin Kabir, Even Oldridge. "Transformers with Multi-Modal Features and Post-Fusion Context for e-Commerce Session-Based Recommendation", In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval July 2021SIGIR eCom'21, Montreal, Canada, July 15, 2021. pp. 143-153. https://doi.org/10.48550/arXiv.2107.05124

Chapter 6
Learning to Rank in Session-Based Recommender Systems

Abstract Today, our daily activities are increasingly dependent on data-oriented systems. A new trend emerged based on machine learning techniques to rank the results in information retrieval and recommender systems automatically called learning to rank (LtR). Two main important subsets of LtR systems include ranking creation and ranking aggregation. This chapter of the book discussed different models of LtR in information retrieval, recommender systems, and session-based recommender systems.

Keywords Learning to rank · LtR · Recommender systems · Information retrieval · Session-based recommender systems · Ranking creation · Ranking aggregation

6.1 Introduction

Due to the increasing expansion of data-driven platforms such as social networks in our daily lives, different types of information retrieval systems play a significant role in organizing our activities. Information retrieval systems have access to information sources and help users make different decisions. For this reason, the approaches of ranking items, prioritizing, and presenting them to the users are important and effective. In recent years, a field called learning to rank (LtR) has emerged based on a combination of machine learning and information retrieval. Learning to rank uses machine learning techniques to rank the results, which is applied in various fields such as document retrieval, entity search, personalized search, collaborative filtering, document summarization, meta-search, question answering, etc. The LtR approaches are classified into ranking creation and ranking aggregation [1]. Ranking creation is making a ranking list of objects using their attributes, while ranking aggregation builds a ranking list of objects using multiple ranking list techniques. Various methods in these two fields usually employ supervised, semi-supervised, or unsupervised machine learning approaches. Learning to rank, specially ranking creation, has recently been intensively studied.

Information retrieval systems, specially search engines and recommender systems based on information sources, help users' decision-making process. Various approaches have been proposed to achieve a better quality of recommender systems

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2_6

and improve their ranking performance. Creating a high-quality ranking list is essential for recommender systems, whose final goal is to recommend a prioritized list of the suggested items to users. Although deep learning models have widely shown promising performance in recommender systems, little effort has been made to investigate learning to rank in these systems [2].

In this chapter of the book, first, a brief overview of the fundamental of learning to rank models and commonly used datasets in various research in this field will be discussed. Then in Sects. 6.3 and 6.4, the various approaches to rank creation and rank aggregation are discussed and reviewed. Section 6.5 discusses and analyzes the results and the existing issues related to the learning to rank models in session-based recommender systems and provides guidelines for future research in this scope.

6.2 Fundamentals

Learning to rank (LtR) is a subfield of machine learning that considers methods and theories for automatically creating a data model for a ranking problem [3]. In other words, learning to rank is a machine learning technique that automatically creates a ranking function for specific objects.

LtR, as a supervised learning-based method, has been widely used in IR (information retrieval) to generate ranking functions based on training datasets. The ranking function is used to rank documents retrieved in response to a user query. Figure 6.1 shows the high-level process of the LtR that most information retrieval systems follow. For this purpose, the training set made of query-document pairs is given as input to the machine learning algorithm. A ranking model or ranking function is created based on the trained model and then used to rank search results for user queries. The ranking model can also be used in the testing phase to measure

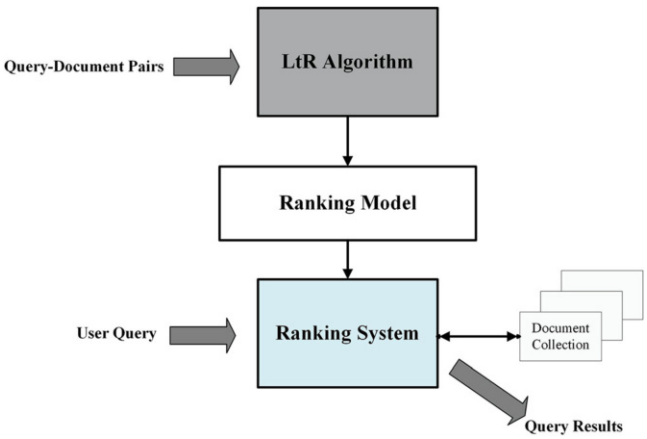

Fig. 6.1 High-level process of LtR in information retrieval systems

the predictive performance of the ranking algorithm on the test dataset. Finally, the ranking system produces an ordered list of documents retrieved from the document collection (document repository) in response to the user's search query.

Two main important subsets of LtR include ranking creation and ranking aggregation, each described and formulated separately along with their learning methods in subsequent sections.

6.2.1 Ranking Creation

The purpose of ranking is to create a ranking list of recommendations based on the features of recommendations and requests so that better recommendations are ranked higher. Learning methods in ranking creation are related to the automatic construction of the ranking model using machine learning techniques. In previous information retrieval systems, learning was not performed to obtain a ranking model to sort documents based on queries. As an example, assuming query q and document d in the BM25 model, the ranking model $f(q, d)$ is presented with a conditional probability distribution such as $P(r|q, d)$, where the value of r is equal to zero or one and indicates the unrelatedness or relatedness of the document, respectively. In the language model for IR (LMIR) [4], the conditional probability distribution $P(q|d)$ represents the ranking model. The computation of probabilistic models is done through observed words in documents and queries and is independent of learning.

After that, a new trend emerged in information retrieval that used machine learning techniques to create ranking models automatically. In the information retrieval scope, a lot of data present the relations and can be used in automatically creating the ranking model. It also provides a new opportunity to automatically create a low-cost ranking model by extracting training data from search logs. Therefore, learning to rank has become one of the effective technologies for modern Web search engines [1]. Figure 6.2 shows the framework of learning to rank [3].

As shown in Fig. 6.2, since learning to rank is a type of supervised learning, it needs a training set. Generating a training set is similar to creating a test set for evaluation purposes. In this framework, a set of n training queries are represented by $q_i (i = 1, \ldots n)$ and documents related to them, represented by feature vectors by $x^{(i)} = \left\{ x_j^{(i)} \right\}_{j=1}^{m\,(i)}$. $m\,(i)$ is the number of documents related to q_i query. Ground truth labels are also specified by $y = \left\{ y_j \right\}_{j=1}^{m}$. Then, a special learning algorithm is employed to learn the ranking model so that the output of the ranking model can predict the ground truth labels in the training set as accurately as possible and according to the loss function. In the test phase, when a new query is entered, the built and trained model in the previous phase is used to sort the documents and return the corresponding ranking list to the user.

The field of ranking creation includes four main issues: training and testing processes, creating high-quality training data, feature construction, and evaluation.

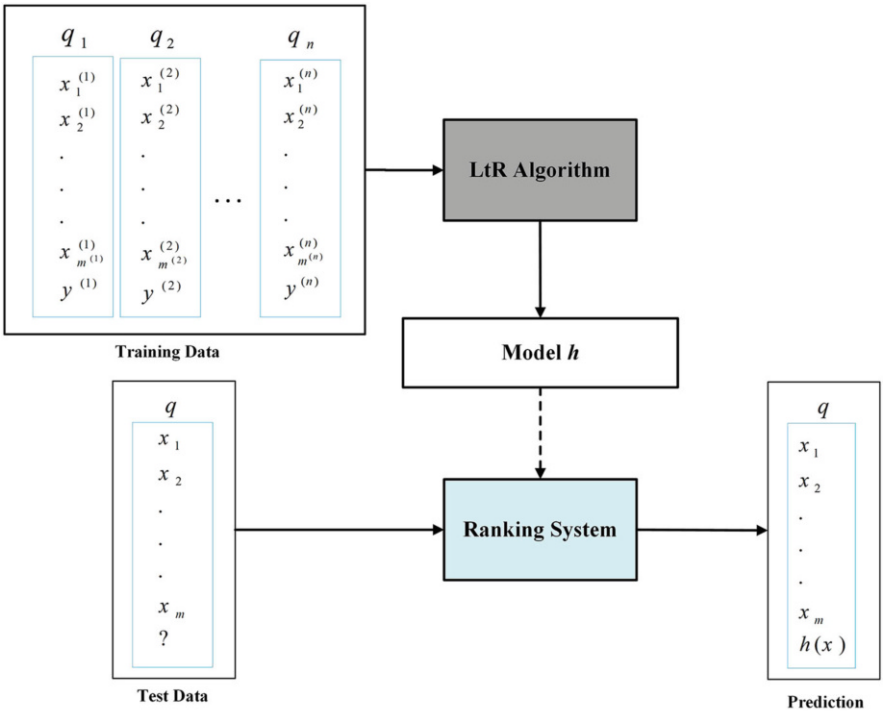

Fig. 6.2 Learning to rank framework [3]

If ranking creation is performed using a supervised approach, training and test data are the crucial components. For example, in information retrieval, sets of queries and documents are considered training data, which include the degree of relevance between each query and the documents. However, in real-world scenarios, this kind of data can be difficult to obtain because ranking lists must contain average judgments of users about the relevance of documents to queries. Typically, there are two common methods of training data creation. The first labeling method is by human users, which is widely used in various fields of information retrieval. Another method is to extract data through clicks. Click-through data in a Web search engine records the user clicks on documents after submitting a query. Click-through data presents implicit feedback about users' relevance and is therefore useful for relevance judgments. It is worth mentioning that the ranking model is actually defined as a function of the feature vector based on the document and the query. This is why the ranking model is generalizable, and even if it is trained on a small amount of data, it can be extended to use for any other data. Like other machine learning tasks, learning performance strongly depends on the features' effectiveness. Therefore, the method for feature construction is critical. Finally, performance evaluation of a ranking model is performed by comparing the ranking list of the model's output and the ranking list provided as the ground truth.

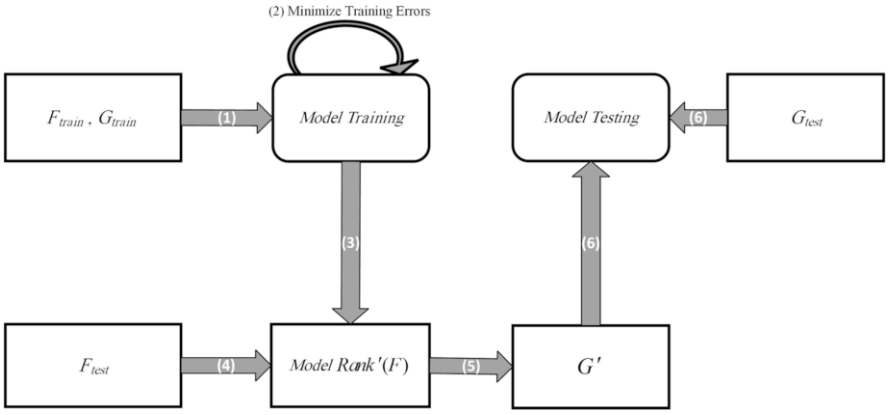

Fig. 6.3 The supervised LtR process

The supervised LtR process is depicted in Fig. 6.3. LtR process includes datasets for training and testing purposes, which are indicated by D_{train} and D_{test}, respectively. D_{train} is used to train an LtR model, and its purpose of this training is to minimize the prediction error on G_{train} based on the ranking function $\acute{Rank}(F)$. G is a set of score features and ground truth for supervised learning. This process is usually done by minimizing the sum of the individual errors $\acute{Rank}$ between the ground truth G and its prediction $G^{'}$ for D_{train}. To evaluate the performance of the $\acute{Rank}$ model, we apply it to D_{test} and then compare the ground truth scores and predictions. If the ranking predictions are considered accurate enough, then the previous test result has succeeded. So $\acute{Rank}$ considers a new set of candidates to predict their scores $\acute{G} = \acute{Rank}(F)$, and candidates are ranked based on these predictions.

In step (1) of Fig. 6.3, the training data D_{train}, consisting of tuples *(F, G)*, is given as input to a learning to rank algorithm, which in step (2), a ranking function $\acute{\text{Rank}}(F)$ is trained. This is done by minimizing the errors $\acute{\text{Rank}}(F)$ when predicting the scores $G^{'}$ for D_{train} in step (3). The prediction accuracy of the model is evaluated in (4); for this purpose, the F_{test} features from D_{test} are taken as input for $\acute{\text{Rank}}$. In step (5), the predicted scores for $G^{'}$ are calculated, and then $G^{'}$ is compared with the ground truth G_{test} in (6).

It should be noted that the approaches presented in the field of ranking creation based on learning methods are divided into three categories: pointwise, pairwise, and listwise. Pointwise and pairwise approaches transform the ranking problem into classification, regression, and ordinal regression. The listwise approach takes the ranking lists of objects as examples for learning and learns the ranking model based on the ranking lists. The main differences between the approaches are based on the loss functions used.

6.2.2 Ranking Aggregation

The purpose of ranking aggregation is to combine multiple rankings into a single ranking, which is better than any of the original rankings in terms of evaluation metrics. Learning in ranking aggregation involves building a ranking model for ranking aggregation using machine learning techniques. For example, in meta-search, the query from the user is sent to several search systems, and the ranking lists from the search systems are combined and presented to the user in a ranking list. Because rankings from individual search engines may not be accurate enough, meta-search actually takes the majority of votes over search rankings. The question is how to effectively perform majority voting. Here, individual search engine rankings are called base rankings, and meta-search rankings are called final rankings. LtR aggregation can be done through unsupervised or supervised learning approaches. In previous information retrieval methods, ranking aggregation was usually based on unsupervised learning. Recently, supervised methods for ranking aggregation have also been proposed. In supervised LtR aggregation, the training data includes queries, their associated documents, and the basic rankings on the documents, as well as the corresponding final rankings. Test data include queries, related documents, and basic rankings of documents. Finally, the evaluation metrics in the ranking aggregation are based on how to present the ground truth. It can be any standard measure in LtR systems [1].

Another type of LtR classification method includes feature-based and discriminative methods. Feature-based methods, called label learning methods in machine learning, represent all the available documents with feature vectors that reflect the relevance of the documents to the query. Conventional features used in learning to rank are the frequency of query terms in the document, the outputs of the BM25 model, and the PageRank model. These features can be obtained from the indexes of a search engine. LtR aggregation methods can combine several features, and by including the output of the model as one of the dimensions of the features, it is able to apply any new development in the information retrieval model. This capability enables real search engines to use multiple features to detect the required complex information of Web users. Learning to rank techniques based on discriminative methods have an automatic learning process based on training data. These methods are often needed for real search engines because these search engines receive much feedback from users and usage reports every day. Therefore, automatic learning from feedback and continuous improvement of the ranking are critical. In addition, discriminative methods are utilized to combine different types of features without the need to define a probabilistic framework to provide entities and predict accuracy.

All learning to rank approaches generally learn their ranking functions by minimizing some loss functions. In recommender systems, a top N list is usually generated to be displayed to users. Hence, ranking this list is crucial for both the quality of recommendations and user satisfaction. Indeed, instead of focusing on recommendations as a rank prediction problem in recommender systems, looking at how items are ranked is more reasonable. The item most related to the user should be

Table 6.1 Reviewed studies in LtR scope (information retrieval and recommender systems)

Learning to rank model		Reference		Ranking model	
		Information retrieval	Recommender system	Information retrieval	Recommender system
Rank creation	Pointwise	[7–17]	[18–22]	SubsetRanking [7], OPRF [8], McRank [11], PRanking [13], DPG-FBE [16]	CPL-mg [22]
	Pairwise	[23–34]	[35–42]	RankNet [26], LambdaRank [23], FRank [28], Ranking SVM [29], GBRank [30], SortNet [31], SSRankBoost [32]	PRM [41], BPR [36], TOP1[37], CPLR [38], PDLR [35], PLtR-N [40]
	Listwise	[43–50]	[51–54]	AdaRank [45], ListNet [46], PiRank [47], SetRank [48], FastAP [49], DLCM [50], RaMBO [44]	TOP1-max [52], BPR-max [52], Do-Rank [53]
	Hybrid	[55–57]	[2, 58–60]	LambdaRank [56], LambdaMART [56], IESR-Rank [57], IESVM-Rank [57]	DeepRank [2]
Rank aggregation		[34, 61–68]	[69–73]	Cranking [63], v-ManX [65], RABF [68]	ERA [70]

at the top of the list of recommendations. Defining relevance eliminates the need to predict rankings. You do not need to know how many users rate something; what matters is that the user at least likes that item more than anything else available. It is worth noting that the item catalog may not contain an item the user likes, but in that case, the recommender system still wants to provide a list of the best available items it has [5]. Ranking models and algorithms help recommender systems arrange the items of the list of recommendations in the most optimal state possible [6].

Recommender systems that learn a ranking model based on the preference scores of individual items are considered pointwise ranking methods. Recommender systems using pairwise learning to rank can consider each user's preferences for a pair of items, and finally, listwise learning to rank systems can consider each user's list preferences for a list of items (usually ranked) model.

Table 6.1 presents the list of studies according to the type of ranking (creation/aggregation) and their application (IR/RS).

The diagram in Fig. 6.4 shows the percentage of each technique used in the discussed research.

According to the diagram in Fig. 6.4, a large percentage of the research reviewed is related to methods in ranking creation which are devoted to the automatic construction of the ranking model using machine learning techniques.

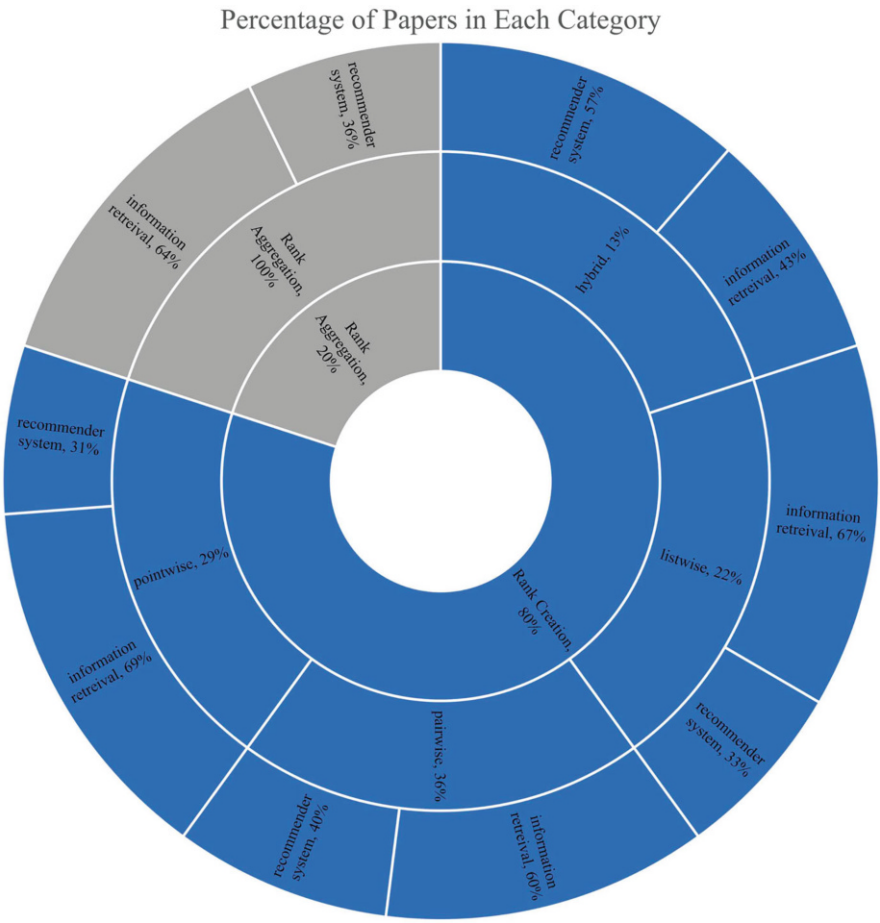

Fig. 6.4 Percentage of each type of ranking creation and ranking aggregation methods in IR/RS scope

6.2.3 Datasets

To evaluate learning to rank systems, several datasets have been published publicly. This dataset contains thousands of annotated queries, hundreds of semantic features, and millions of user sessions, which have been widely used in learning to rank research. Publicly available learning to rank datasets can be roughly classified using synthetic or real user feedback. Both are widely used in the empirical study of unbiased learning to rank algorithms.

- Microsoft LETOR [74]: It uses the Gov2 Web page collection (~25 million pages) and two sets of queries from the Million Query track derived from TREC 2007 and TREC 2008. These two query sets are called MQ2007 and

MQ2008. In MQ2007, there are 1692 queries with 65,323 labeled documents; in MQ2008, there are about 784 queries with 14,384 labeled documents.

- Yahoo! LETOR [75]: It is one of the largest datasets for public LtR that have used commercial English search engines. In total, it contains 29,921 queries with 710,000 documents. Each query and document pair has a five-level relevance judgment and 700 features selected by a separate feature selection phase.
- Tiangong-ULTR [76]: The dataset was collected using real-world user click data sampled from Sogou.com search sessions. For this purpose, 3449 queries written by real search engine users were randomly sampled, and the top 10 results were collected from a 2-week search report. After cleaning, the dataset has 333,813 documents, 71,106 ranked lists, and 3,268,177 anonymized search sessions with clicks.
- Istella-S [77]: It contains 33,000 queries and 3,408,000 documents (approximately 103 documents per query) sampled from an Italian commercial search engine. Each query-document pair is represented by 220 features and annotated with five-level relevance judgments.
- Baidu-ULTR [78]: The dataset consists of two parts: (1) large-scale Web search sessions and (2) expert annotation dataset. The first consisted of 383,429,526 queries and 1,287,710,306 documents randomly sampled from Baidu search engine search sessions in April 2022. Most sessions contain less than ten candidate documents with page presentation features and user behaviors in the current query. The latter is also randomly sampled from the monthly collected search sessions of the Baidu search engine, and the relevance of each document to the query is judged by expert annotators, which includes five tags.

6.3 Ranking Creation

Ranking creation is the ranking of recommended lists based on the features of the recommendations so that the more relevant recommendations are ranked at a higher level. In this section, we will discuss the main ranking creation models, which are generally classified into pointwise, pairwise, and listwise in the two fields of information retrieval and recommender systems.

6.3.1 Pointwise Methods

In pointwise methods, the ranking problem becomes a classification, regression, or ordinal regression problem, and existing classification, regression, or ordinal regression methods can be used to solve it. These types of methods take training data as input and ignore group structures. Training data is transformed into supervised learning data so that existing methods can be used to perform learning. When a class label, a real number, and a score label are considered for the data in the dataset,

the problem becomes classification, regression, and ordinal regression, respectively. Assuming that the output of the learned model is real numbers, this model can be used to rank documents when a query is given (sort documents according to the scores given by the model). The loss function is defined in pointwise methods on a single object. In fact, the total loss in pointwise methods is calculated as the sum of the defined losses for each document as the distance between the predicted score and the real score.

In the field of information retrieval systems, four basic components are formally defined to describe pointwise methods: input, output, hypothesis, and loss function [3]:

- The input space contains the feature vectors of each document.
- The output space contains the degree of relevance of each document. The ground truth label in the output space is defined as follows: if the judgment is directly assumed as the degree of relevance l_j, the ground truth label for the document x_j is defined as $y_j = l_j$, but if the judgment is defined as the total order of π_l, the ground truth label can be obtained using a mapping function. It is worth mentioning that if the judgment is given as a pairwise preference $l_{u,v}$, it is not easy to use it to generate the ground truth label.
- The hypothesis space contains functions that take the feature vector of a document as input and predict the degree of relevance of the document. Usually, such a function is called the scoring function f. All documents can be ordered based on the scoring function, and the final ranking list can be delivered.
- The loss function measures the accurate prediction of the ground truth label for each document. In different pointwise ranking algorithms, ranking is modeled as regression, classification, and ordinal regression. Therefore, regression loss, classification loss, and ordinal regression loss are utilized as loss functions.

As mentioned above, based on different machine learning techniques, the pointwise approach can be divided into three categories: classification-based algorithms, regression-based algorithms, and ordinal regression-based algorithms [3]:

- The ranking problem can be considered as a classification model. Classification is a supervised learning problem in which the predicted target variable is discrete. When ranking is modeled as a classification, the degree of relevance given to a document is considered a class label. In classification-based algorithms, the output space contains unordered categories. Some of these approaches are based on binary classification, and some others are based on multi-class classification. Approaches related to binary classification are based on support vector machines (SVM) [9], logistic regression [10], etc. SVM-based methods have been noticed due to their ability to automatically learn arbitrary features, fewer assumptions, and expressiveness [9]. Logistic regression is a popular classification technique used to perform binary classification for ranking. Approaches based on multi-class classification use boosting tree-based method techniques [11], association rule mining [12], etc.

Table 6.2 Learning methods in pointwise approaches

	Regression	Classification	Ordinal regression
Input	Single document x_j		
Output	Real number y_j	Non-ordered category y_j	Ordered category y_j
Hypothesis	$f(x_j)$	Classifier on ($f(x_j)$)	$f(x_j)$ + thresholding
Loss function	Regression loss $L = (f; x_j, y_j)$	Classification loss	Ordinal regression loss

- For regression-based algorithms, by considering the degree of relevance as real numbers, the ranking problem is reduced to a regression problem. Regression is also a supervised learning problem where the predicted target variable is continuous. When ranking is modeled as regression, the degree of relevance given to a document is considered a continuous variable, and the ranking function is trained by minimizing the loss of the training set. In these methods, the output space contains points associated with real values. In regression-based approaches, various types of regression have been used, such as weighted regression [7], polynomial regression [8], etc.
- Ordinal regression considers the ordinal relationship between ground truth labels when learning a ranking model. For algorithms based on ordinal regression, the output space consists of ordered categories. Documents with their ground truth labels are considered in the training set as random independent variables with the same distribution resulting from the multiplication of the input spaces [3]. If the number of sorted categories is considered equal to 2 in this type of approach, the problem is reduced to a binary classification. For this reason, ordinal regression-based techniques strongly correlate with classification-based algorithms. These types of approaches use perceptron-based ranking methods [13, 14], ranking with large margin principles [15], and loss functions based on threshold [17].

Table 6.2 shows the learning methods of pointwise approaches, separated by input, output, hypothesis, and related loss function for regression, classification, and ordinal regression approaches.

Pointwise approaches in recommender systems create a score for each item and then rank the items based on that score. Then, the recommendations are presented to users based on this rating. The difference between score prediction and ranking is that with ranking, the score of an item is important when it represents the correct position of that item in the ranking. In Sect. 6.3.1.1, the pointwise methods in the field of information retrieval and, in Sect. 6.3.1.2, the pointwise methods in the field of recommender systems are discussed and reviewed.

6.3.1.1 Pointwise Methods in Information Retrieval

One of the fundamental methods for pointwise LtR is the method provided by Cossock et al., which models the ranking problem based on the regression technique [7]. The set of m documents related to query q is represented by $x = \{x_j\}_{j=1}^{m}$, and the

set of $y = \{y_j\}_{j=1}^{m}$ represents the ground truth labels of the documents sorted by multiple ordered categories. The scoring function f is used to rank these documents, and the loss function is calculated using Eq. (6.1):

$$L(f; x_j, y_j) = (y_j - f(x_j))^2 \tag{6.1}$$

The results of this method show that there is no loss if and only if the output of the scoring function $f(x_j)$ is exactly equal to the label y_j. Otherwise, the loss value increases to the power of 2. In other words, for a related document, the loss will be zero only if the scoring function can output exactly y_j. The specific loss of this function can be the upper limit of the ranking error based on the nDCG metric. In the case of high regression loss, the corresponding ranking is optimal as long as the relative orders between the predictions of $f(x_j)$ correspond to the ground truth labels. As a result, it is expected that the squared loss function is a loose bound of the nDCG-based ranking error.

With the aim that ranking errors based on DCG is limited by multi-class classification errors, Li et al. proposed multi-class classification, the McRank algorithm [11]. They learn a classification model and use it to obtain the membership probability of each object. Then, they calculate the expected scores of the objects and use them for ranking purposes. Class probabilities are learned using the gradient boosting tree algorithm.

The loss function used to learn the classifier in McRank is according to Eq. (6.2):

$$L(\bar{y}_j, y_j) = I_{\{y_j \neq \bar{y}_j\}} \tag{6.2}$$

The surrogate loss function is defined according to Eq. (6.3), and the boosting tree algorithm is used to minimize the error.

$$L_{\varphi}(y_j, \bar{y}_j) = \sum_{j=1}^{m} \sum_{k=1}^{K} - \log P(\bar{y}_j = k)\, I_{\{y_j = k\}} \tag{6.3}$$

In Eq. (6.3), $P(\bar{y}_j = k)$ is defined using the logistic function, which is calculated using Eq. (6.4):

$$P(\bar{y}_j = k) = \frac{e^{F_k(x_j, w)}}{\sum_{s=1}^{K} e^{F_s(x_j, w)}} \tag{6.4}$$

In Eq. (6.4), the expression $F_k(x_j, w)$ shows the degree of belonging of the document x_j to category k. Then, classification results are converted into ranking scores. The output of the classifier is converted to probability using Eq. (6.4). Then,

the weighted combination of Eq. (6.5) is used to determine the final ranking score of a document.

$$f(x_j) = \sum_{k=1}^{K} g(k).\ P(\bar{y}_j = k) \tag{6.5}$$

Another well-known pointwise method based on ordinal regression called PRanking has been proposed by Crammer et al. [13]. The goal of this algorithm is to find the path defined by the parameter vector w after presenting the documents on it. Then, thresholds can be easily used to identify the documents sorted into different categories. This goal is achieved using an iterative learning process. In iteration t, the learning algorithm receives an instance x_j associated with query q. The algorithm predicts $\bar{y}_j$ according to Eq. (6.6) and receives the ground truth label y_j.

$$\bar{y}_j = \arg\min_k \left\{ w^T x_j - b_k < 0 \right\} \tag{6.6}$$

If the algorithm makes a mistake and there is at least one threshold indexed by k, the value of $w^T x_j$ is in the wrong side from b_k. To correct the error, values of $w^T x_j$ and b_k should be directed toward each other on the same side. After that, the model parameter w is set by $w = w + x_j$, same as in many perceptron-based algorithms. This process is repeated until the training process converges.

Hu et al. considered item ranking in search sessions on e-commerce platforms such as Amazon and Taobao, which is a multi-step decision-making problem [16]. For greater correlation between different ranking steps, the authors presented a reinforcement learning method to learn the optimal ranking policy in which expected accumulative rewards are maximized in one search session. For this purpose, the concept of the search session Markov decision process (SSMDP) is formally defined to formulate the multi-step ranking problem by identifying the state space, reward function, and state transition function. In addition, a new algorithm named Deterministic Policy Gradient with Full Backup Estimation (DPG-FBE) is proposed, which is used for the problem of high reward variance and unbalanced reward distribution of SSMDP.

6.3.1.2 Pointwise Methods in Recommender Systems

One of the pointwise methods presented in the field of video recommender systems, which is a large-scale multi-objective ranking system, recommends the next video to watch on a commercial video-sharing platform [19]. This method is a combination of classification and regression approaches that have multiple objectives. The ranking problem in a video recommender system has several challenges. They usually have different or conflicting goals; high-rated videos may be recommended as well as videos shared with the viewer's friends. Or, there is often an implicit bias in the system. For example, a user may have clicked and watched a video just because it

was rated high, not because the user liked it the most. Therefore, models trained using data generated from the current system will be biased, causing a feedback loop effect [21].

Zhao et al. have employed user behaviors as training labels [19]. Since users can behave differently from what is recommended, the ranking system is designed to support multiple objectives. This method divides goals into two categories: (1) participation objectives, such as user clicks and the degree of participation with recommended videos, and (2) satisfaction objectives, such as a user liking a YouTube video and rating a recommendation. The prediction of behaviors related to interaction goals is formulated into two tasks: binary classification for behaviors such as clicks and regression for behaviors related to time spent watching. Similarly, predicting user satisfaction behaviors are described as binary categories or regressions. For example, behaviors such as clicking like for a video are described as a binary classification task, and behaviors such as rating as a regression task. The binary classification is calculated from the cross-entropy loss function, and the squared loss is calculated for regression. The proposed framework of Zhao et al. has been shown in Fig. 6.5.

Zhu et al. proposed a new framework called CPL (Combined framework of Pointwise prediction and LTR) in which pointwise prediction and LTR are combined to improve the performance of top N recommendations [22]. In fact, there may be a problem of overfitting in pointwise prediction, while LTR is more prone to higher variance. Both problems can be improved using the hybrid model presented by the authors. For this purpose, they propose a special implementation of CPL called CPLmg, which is a combination of FSLIM and GAPfm models. FSLIM is extended from SLIM using matrix factorization (MF) methods. SLIM directly learns a similarity matrix from the data. Factorized SLIM is a new version of SLIM that incorporates the idea of traditional matrix factorization (MF) methods. GAPfm is a list LTR method that directly optimizes a smoothed approximation of the GAP metric. GAP extends average precision to cases of multi-graded relevance and inherits the most important feature of the AP metric, which ensures that errors in recommended items at the top of the list are penalized more than errors at the bottom of the list.

The authors train FSLIM and GAPfm using the multi-task learning approach. As shown in Fig. 6.6, FSLIM and GAPfm are iteratively updated based on their respective objective functions with shared underlying variables, latent factor matrices P and Q, and learned item similarity matrix W. The matrices P and Q are jointly updated by FSLIM and GAPfm in each joint training round, and the item similarity matrix W helps GAPfm to extract potential positive samples.

To learn and estimate several types of user behavior, Ma et al. used MMoE [18] to automatically learn parameters and share potentially conflicting objectives. MMoE is a soft parameter-sharing model developed to model task relations. It adapts the mixture-of-experts (MoE) structure for multi-task learning by sharing experts in all tasks while also having a trained gating network for each task. The key idea is to replace the shared ReLU layer with the MoE layer and add a separate gating network for each task.

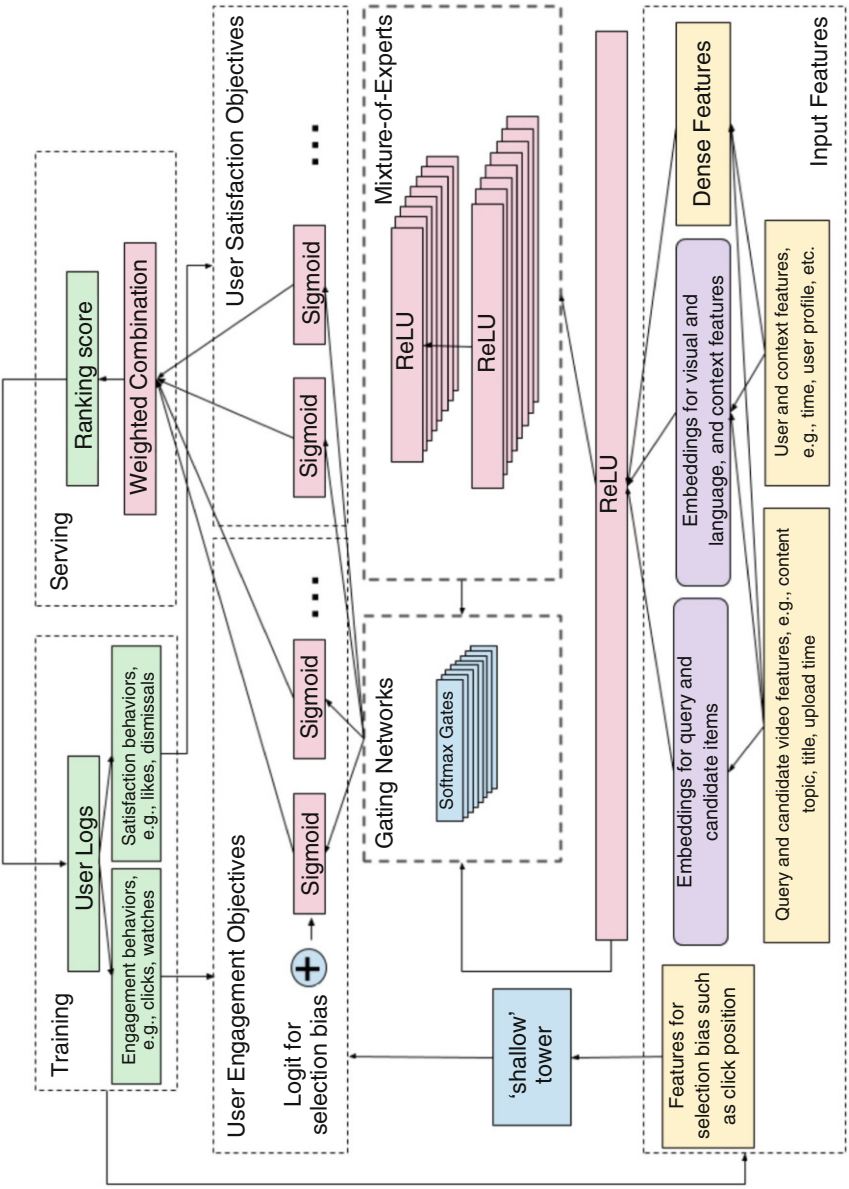

Fig. 6.5 The framework of end-to-end ranking system [19]

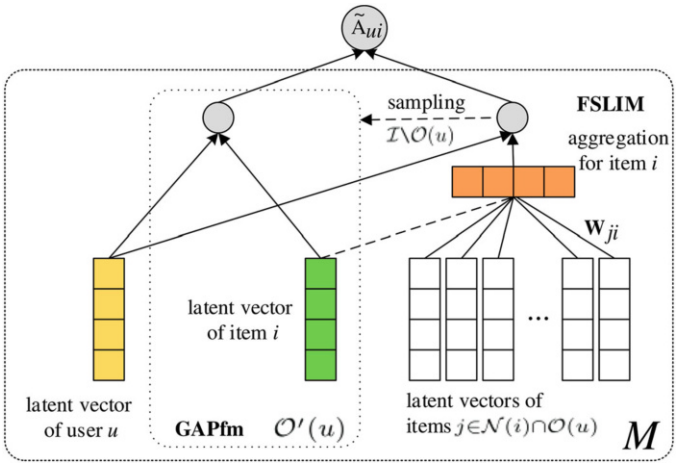

Fig. 6.6 The CPLmg framework [22]

In [20], Tang et al. presented a pointwise ranking method that considers multiple objectives for modeling different user behaviors, such as clicking, sharing, and commenting in the ranking system. In the offline training process, the ranking model is trained based on user actions extracted from user logs. After each online request, the ranking model provides predictions for each task, and then the weighted multiplication-based ranking module combines these predicted scores through a combination function. Among all tasks, VCR (view completion ratio) and VTR (view-through rate) are two important objectives that model the key online metrics of view count and watch time, respectively. VCR is a regression-based task trained on MSE, and VTR is a binary classification task trained on cross-entropy. The patterns between VCR and VTR are complicated, and the SEESAW phenomenon exists between them. The SEESAW phenomenon is defined as the improvement of one task often leading to a decrease in the performance of another task.

In this chapter, a progressive layered extraction (PLE) model is proposed to deal with the SEESAW phenomenon and negative transfer. The key idea of PLE is as follows. First, it explicitly separates common and task-specific experts to avoid harmful parameter interference. Second, multi-level experts and gating networks are introduced to incorporate more abstract representations. Finally, it adopts a novel progressive separation routing to model interactions between experts and achieves more efficient knowledge transfer between complex related tasks.

6.3.2 Pairwise Methods

The pairwise approach does not focus on accurately predicting the degree of relevance of each document but rather on the relative order between two documents.

Table 6.3 Components of pairwise approaches

	Pairwise methods
Input	Document pair (x_u, x_v)
Output	Preference $y_{u,\,v}$
Hypothesis	Ranking function $f\left(\vec{X}\right)$ $2.I_{\{f(x_u)>f(x_v)\}} - 1$
Loss function	*Pairwise classification/regression loss* $L\,(f\,;\, x_u, x_v, y_{u,\,v})$

In this sense, it is closer to the concept of "ranking" than the pointwise approach. In the pairwise approach, ranking is usually reduced to classification over pairs of documents, determining which document is preferred in a pair. Therefore, the goal of learning is to minimize the number of unclassified document pairs. This classification differs from the classification in the pointwise approach because it operates on both documents under review. In the field of information retrieval systems, four essential components are formally defined to describe pairwise methods: input, output, hypothesis, and loss function [3].

- The input space contains pairs of documents, both represented by feature vectors.
- The output space contains the pairwise preferences between each pair of documents, which take the values $\{+1,-1\}$. Different types of judgments can be converted into ground truth labels in terms of pairwise priorities:
  - If the judgment is assumed as the degree of relevance l_j, then the pairwise preference for (x_u, x_v) can be defined as $y_{u,v} = 2.I_{\{l_u > l_v\}} - 1$.
  - If the judgment is assumed directly as a pairwise preference, it is simply set as $y_{u,\,v} = l_{u,\,v}$.
  - If the judgment is assumed to be the total order of π_l, it can be defined as $y_{u,v} = 2.I_{\{\pi_l(u) < \pi_l(v)\}} - 1$.
- The hypothesis space consists of two-variable function h, which takes a pair of documents as input and outputs the relative order between them. Some pairwise ranking algorithms define their hypotheses directly, and others define this hypothesis with a scoring function f according to Eq. (6.7):

$$h(x_u, x_v) = 2.I_{\{f(x_u)>f(x_v)\}} - 1 \tag{6.7}$$

- The loss function measures the inconsistency between $h(x_u, x_v)$ and the ground truth label, $y_{u,\,v}$. In many pairwise ranking algorithms, the ranking is modeled as a pairwise classification, and the loss of the corresponding classification on a pair of documents is used as a loss function. The scoring function f uses the classification loss function based on the difference $(f(x_u), f(x_v))$ instead of the value $h(x_u, x_v)$.

Table 6.3 shows the components of pairwise approaches, separated by input, output, hypothesis, and related loss function.

Several recommender systems use pairwise methods to rank recommended items. In particular, session-based recommender systems that are based on sequential data and usually apply recurrent neural networks mostly use pairwise ranking loss functions [52]. Since deep learning methods need to propagate gradients in several layers to optimize model parameters, the quality of these gradients caused by the loss function affects the optimization quality and model parameters. Additionally, output spaces with several items present unique challenges that must also be considered when developing an appropriate ranking loss function.

The basic approaches to pairwise LtR have different types. Some of them are based on neural networks [34], perceptrons [31], boosting [33], SVM [29], and other machine learning methods [28, 30]. In Sect. 6.3.2.1, the pairwise methods in the field of information retrieval and, in Sect. 6.3.2.2, the pairwise methods in the field of recommender systems are discussed and reviewed.

6.3.2.1 Pairwise Methods in Information Retrieval

In [26], one of the first pairwise methods for learning to rank was proposed by Burges et al. using neural networks, which is called RankNet. RankNet selects the cross-entropy function as the loss function for learning and considers probability in the training data for each pair of relevant objects. If you consider two documents x_u, x_v associated with q, $\bar{P}_{u,v}$ is calculated based on their ground truth labels. In this case, if $y_{u,\,v} = 1$, then $\bar{P}_{u,v} = 1$; otherwise, $\bar{P}_{u,v} = 0$. Then, the modeled probability $P_{u,\,v}$ based on the difference between the scores of these two documents is defined by the scoring function according to Eq. (6.8):

$$P_{u,v}(f) = \frac{\exp(f(x_u) - f(x_v))}{1 + \exp(f(x_u) - f(x_v))} \tag{6.8}$$

Cross-entropy between the final probability and the modeled probability is used as a loss function, which is briefly mentioned in Eq. (6.9):

$$L\left(f; x_u, x_v, y_{u,v}\right) = -\bar{P}_{u,v} \log P_{u,v}(f) - (1 - \bar{P}_{u,v}) \log(1 - P_{u,v}(f)) \tag{6.9}$$

Cross-entropy loss is an upper bound of the 0-1 pairwise loss function, which is defined in Eq. (6.10):

$$L_{0-1}\left(f; x_u, x_v, y_{u,v}\right) = \begin{cases} 1 & y_{u,v}(f(x_u) - f(x_v)) < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6.10}$$

Then a neural network is used as a model and stochastic gradient descent as an optimization algorithm to learn the scoring function f. This neural network consists of two normal layers and an output layer, where the features of a document are entered in the first layer. The second layer consists of several hidden nodes, each

containing a sigmoid transformation, and the network output is the document ranking score.

Burges et al. proposed a nested pairwise ranking method based on RankNet, which iteratively re-ranks documents with higher scores [27]. At each iteration, this approach uses the RankNet algorithm to re-rank a subset of the results. It divides the problem into smaller and simpler parts and creates a new distribution of results to be learned by the algorithm.

In the method presented in [26] by Burges et al., in some cases, cross-entropy loss has a non-zero minimum, which shows that there will always be some loss regardless of the type of model used. This may not match our cognition of a loss function. In addition, the loss is not bounded, which may lead to the dominance of some difficult document pairs in the training process. To deal with these problems, a new loss function called fidelity loss function is proposed in [28], which is defined by Eq. (6.11):

$$L\left(f; x_u, x_v, y_{u,v}\right) = 1 - \sqrt{\bar{P}_{u,v}\, P_{u,v}(f)} - \sqrt{(1-\bar{P}_{u,v})(1-P_{u,v}(f))} \tag{6.11}$$

Fidelity was originally used in quantum physics to measure the difference between two possible states of a quantum. When used to measure the difference between the target probability and the modeled probability, it is $(f(x_u) - f(x_v))$. By comparing loss fidelity with cross-entropy loss, it is clear that fidelity loss is limited between 0 and 1. On the other hand, while cross-entropy loss is convex, fidelity loss becomes non-convex, which makes optimizing such a non-convex objective more difficult. In addition, fidelity loss is no longer an upper bound of 0-1 pairwise loss. In [28], a generative additive model as a ranking function is proposed by Tsai et al., which is similar to the boosting technique to learn the coefficients in the additive model. Specifically, a new weak ranker (i.e., a new feature) is added in each iteration, and the combination coefficient is adjusted by considering the fidelity loss gradient. The learning process converges when adding a new ranking no longer results in a significant reduction.

The ranking evaluation result based on the learning to rank objective function is usually non-continuous and non-differentiable and depends on the ordering. Of course, it should be noted that the sorting function is not continuous and differentiable. Burges et al. have proposed a method called LambdaRank that considers the use of gradient descent to optimize the evaluation result and tries to directly use the gradient function of the evaluation result [23]. In LambdaRank, position-based weights are introduced for the pairwise loss function. In fact, the evaluation measures (based on position) are directly used to define the gradient according to each pair of documents in the training process.

Suppose there are only two relevant documents x_1 and x_2 and NDCG@1 is used as the evaluation metric. In this case, if we can rank x_2 and x_1 at the top of the list and x_1 is ranked higher, we will achieve the maximum NDCG@1. It is obviously easier to move x_1 up, since x_2 will have much less effort. Therefore, we can define (but not calculate) the "gradient" given the rank score x_1 (denoted by $s_1 = f(x_1)$) greater than

that given the rank score x_2 (denoted by $s_2 = f(x_2)$). In other words, we can consider that there is an implicit loss function L in the optimization process, which Eq. (6.12) shows:

$$\frac{\partial L}{\partial s_1} > \frac{\partial L}{\partial s_2} \tag{6.12}$$

The above gradient is called the lambda function, and that is why the algorithm is called LambdaRank. When nDCG is used for training, it provides a special form of lambda function as shown in Eq. (6.13):

$$\lambda = Z_m \frac{2^{y_u} - 2^{y_v}}{1 + \exp(f(x_u) - f(x_v))} \left(\eta(r(x_u)) - \eta(x_v)\right) \tag{6.13}$$

r(.) represents the position of a document in the previous iteration of the training. The scores of each pair of documents x_v and x_u are updated by $+\lambda$ and $-\lambda$, respectively, in each round of optimization.

One of the types of pairwise learning to rank are SVM-based methods [24, 25]. These types of methods use pairwise classification to implement LtR. Suppose there are n queries $\{q_i\}_{i=1}^{n}$, which have a pair of relevant documents $\left(x_u^{(i)}, x_v^{(i)}\right)$, and their ground truth label is $y_{u,v}^{(i)}$; Assuming the use of linear ranking function $f(x) = w^T x$, the Ranking SVM is formulated as Eq. (6.14):

$$\begin{aligned} \min \frac{1}{2}\|w\|^2 + \lambda \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} \xi_{u,v}^{(i)} \\ \text{s.t.} \quad w^T\left(x_u^{(i)} - x_v^{(i)}\right) \geq 1 - \xi_{u,v}^{(i)}, \quad \text{if } y_{u,v}^{(i)} = 1, \\ \xi_{u,v}^{(i)} \geq 0, \quad i = 1, \ldots, n \end{aligned} \tag{6.14}$$

In Eq. (6.14), $\frac{1}{2}\|w\|^2$ controls the complexity of model w. The difference between SVM and Ranking SVM is in the constraints that are constructed from document pairs. The loss function in Ranking SVM is a hinge loss defined on document pairs. For example, for query q, if a document x_u is labeled more relevant than a document x_v, i.e., $y_{u,v}$ is equal to one, if $w^T x_u$ is greater than $w^T x_v$ by margin 1, the loss does not exist. Otherwise, the loss equals to $\xi_{u,v}$. Hinge loss is an upper bound of 0-1 pairwise loss.

One type of research related to providing the correct ranking order of documents for a query describes the process of creating a list of documents as a Markov decision process (MDP). For this purpose, document ranking in search result diversification is modeled as an MDP, in which each time step is related to a ranking position and each action is related to selecting a document for that position. Given a set of labeled training data, a policy gradient is used to learn MDP parameters. In this process, in each training iteration, the policy gradient algorithm first samples a document list as

its training sample, and then the gradient is estimated according to the list weighted by the absolute performance score. When this model is applied to rank documents in IR, gradient estimation in this way has limitations, and it ignores the relative ordering nature of IR ranking and estimates gradients with high variance. To solve this problem, Xu et al. proposed a policy gradient algorithm in which the gradients are determined by a pairwise comparison of two sampled document lists in a query [79]. This algorithm, called Pairwise Policy Gradient (PPG), repeatedly samples pairs of document lists. It estimates gradients by pairwise comparisons and fully updates model parameters.

6.3.2.2 Pairwise Methods in Recommender Systems

In recommender systems, pairwise methods generate a list of personalized recommendations for users and determine users' pairwise preferences and interests between items. In session-based recommender systems where there is no specific profile for users, these methods collect users' preferences according to their pairs of behaviors, and a set of item pair preferences are used to represent each user.

The BPR-MF proposed by Rendle et al. is a pairwise LtR method developed for recommender system scenarios with implicit feedback [36]. BPR-MF is commonly used for matrix completion problems based on long-term user-item interactions. This method is one of the most widespread pairwise approaches for learning to rank items, which has corrected ranking capabilities with acceptable computational complexity. Some methods, such as matrix factorization, cannot be directly applied to session-based recommender systems because the feature vectors are not pre-computed in the new session. However, this problem can be overcome by using the average feature vectors of the items that occurred in the session as the user feature vector. The similarity of the feature vectors between a proposed item and the current session items is averaged and optimized with the following measure:

$$\mathrm{BPR-MF} = \sum_{(u,i,j)\epsilon D_s} \ln \sigma\left(r_{u,i} - r_{u,j}\right) - \lambda_\theta \|\theta\|^2 \tag{6.15}$$

In Eq. (6.15), the ranking $r_{u,\ i}$ for user u and item i is approximated by the dot product of the corresponding rows in W and H matrices. The parameters of the $\Theta = (W, H)$ model are learned by stochastic gradient descent in multiple iterations on dataset D, which consists of triples of the form (u, i, j). (u, i) is a positive feedback pair, and (u, j) is a negative sample. σ is the logistic function, and λ_θ is the regularization parameter that controls the complexity. The optimized measure in objective Eq. (6.15) is to rank the positive sample (u, i) higher than the unobserved sample (u, j). The goal of BPR is to maximize the probability that the target score is higher than the sample score. The loss function related to BPR in session-based recommender systems used by Hidasi et al. in [52] at a certain point of the session is defined as follows:

$$L_s = -\frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \log\left(\sigma\left(\overline{r}_{s,i} - \overline{r}_{s,j}\right)\right) \tag{6.16}$$

In Eq. (6.16), N_s is the number of samples, $\overline{r}_{s,k}$ is the score of item k at a certain point of the session, i is the target item (the next item of the session), and j represents the negative samples.

One of the pairwise loss functions proposed by Hidasi et al. in [37] is the TOP1 function, which consists of two parts. The purpose of the first part is to raise the score of the target item higher than the score of the samples, while the second part aims to reduce the score of the negative samples to zero. The second part acts as a regularizer, but instead of directly limiting the model weights, it penalizes high scores on negative samples. Since all items in a training example act as negative scores, it lowers overall scores. The TOP1 function is calculated using Eq. (6.17):

$$L_{\text{TOP1}} = \frac{1}{N_s} \sum_{j=1}^{N_s} \sigma\left(\overline{r}_{s,j} - \overline{r}_{s,i}\right) + \sigma\left(\overline{r}_{s,j}{}^2\right) \tag{6.17}$$

In the above equation, j times execution is performed on the negative items N_s (non-relevant), and relevant items are identified by i.

Although BPR empirically performs well for recommending based on implicit feedback compared to existing methods, it (1) treats all unobserved feedback as negative, (2) treats all observed feedback as the same, and (3) ignores the influence between users. Items without observed feedback may be interpreted as user's lack of interest, and some observed data may be noisy and biased. To solve these problems, a Collaborative Pairwise Learning to Rank method called CPLR is proposed in [38] by Liu et al., which considers the influence between users on preferences among both items with observed and unobserved feedback. The objective function of this method is presented by Eq. (6.18):

$$\begin{aligned} \text{CPLR} - \text{OPT} := \sum_{u \epsilon U} \Bigg[& \alpha \Bigg(\sum_{i \epsilon P_u} \sum_{t \epsilon C_u} \ln\left(\sigma\left(C_{uit}\left(r_{ui} - r_{ut}\right)\right)\right) \\ & + \beta \sum_{t \epsilon C_u} \sum_{j \epsilon L_u} \ln\left(\sigma\left(C_{utj}\left(r_{ut} - r_{uj}\right)\right)\right) + \gamma \sum_{i \epsilon P_u} \sum_{j \epsilon L_u} \ln\left(\sigma\left(C_{uij}\left(r_{ui} - r_{uj}\right)\right)\right) \Bigg] \\ & - \frac{1}{2} \lambda_\theta \|\theta\|^2 \end{aligned} \tag{6.18}$$

In this function, P_u represents the positive set, that is, the set of items to which user u has given positive feedback. C_u denotes the collaborative set, i.e., the set of items that have been given positive feedback by at least one of user u's neighbors, but not by himself. L_u represents the set of items that neither the user nor his

neighbors have yet given positive feedback. r_{ui} represents the predicted preference of user u about i. C_{uij}, C_{uit}, and C_{utj} are confidence coefficients and are used to set the confidence of pairwise preferences. α, β, and γ are control coefficients and are used to control the type of relative preferences and their weight in the model. λ_θ is the regularization coefficient and prevents overfitting in the learning process. When α is set to zero, it assumes that the positive and collaborative item sets are incomparable. When β is set to zero, it assumes that cases without positive feedback are incomparable, and when γ is set to zero, it tries to use the transitive property of the relative preference order.

Ludewig et al. developed a recommender system approach that can be used for both personalized search and session-based recommendation to rank hotels based on the latest user interactions and metadata on available items [42]. In the area of hotel search and recommendation, the problem of effective recommendations is critical because the users are either not logged in or are the first users. This means that adaptation to the users' goals and preferences can only be based on the recent interactions observed in the users' ongoing browsing session. For this purpose, a combination of Bayesian personalized ranking (BPR) with matrix factorization, Doc2Vec, and gradient boosting decision trees (GBDT) has been used. To determine the desired ranking, the problem is considered a pairwise LtR task, and several features are extracted based on the log data.

A recommender architecture based on a deep neural network called PDLR (Pairwise Deep Learning to Rank) has been proposed by Zhou et al., in which the set of items is divided into two groups of positive and negative samples, respectively [35]. Then, the pairwise comparison between positive and negative samples is performed to learn the preference degree for each user. PDLR generally includes embedding, pairwise interaction, attention-aware ranking, and output layers. Specifically, in the PDLR architecture, the embedding layer can learn the feature representation for users and items through the average pooling operation, and the attention-aware ranking layer can determine the importance of each item to the user.

Mayerl et al. were inspired by other research on learning to rank and used a pairwise model for hit song prediction [80]. This model takes a pair of songs A and B and predicts whether song A is more popular than song B. For this purpose, a neural network model is proposed that takes the audio features of two songs as its input and produces a label as an output that shows whether the first song is more popular than the second song or not.

DU et al. utilized methods of LtR to recommend future events for a group of users [39]. The GERF (Group Event Recommendation Framework) analyzes various contextual influences on the user's attendance at events and extracts the user's preference for the event by considering all contextual influences. Then, the preference scores of users in a group are considered learning to rank features for group preference modeling. In addition, a fast pairwise LtR algorithm, Bayesian group ranking, is proposed to learn the ranking model for each group.

Two shared memory lock-free parallel SGD schemes are developed by Yagci et al. for personalized paired LtR to improve its scalability [40]. The authors first adapted a block partitioning approach to these settings and proposed the PLtR-B

(Parallel pairwise LtR with Block partitioning) algorithm. They have shown that the no-partitioning approach can be applied to paired LtR such as point LtR, and they have presented the PLtR-N (Parallel pairwise LtR with No partitioning) algorithm. For shared memory lock-free parallelization of matrix completion, a block partitioning idea divides a user item preference matrix into multiple sets of non-overlapping ideal chunks. Then, each processing unit updates a chunk in each set, thereby enabling parallel processing without using locks. This approach was originally developed for personalized pointwise LtR scenarios, where updates are based on the interaction of an item, and in this study, this method is extended to pairwise LtR.

6.3.3 Listwise Methods

Listwise approaches treat the ranking problem more naturally than other learning to rank methods. In particular, it considers ranked lists as samples in learning and prediction. The group structure of the ranking is preserved, and the ranking evaluation measure can be more directly incorporated into the loss functions. Listwise approaches take training data as inputs and view labeled data $\left(x_{i,1}, y_{i,1}\right), \ldots, \left(x_{i,n_i}, y_{i,n_i}\right)$ associated with q_i as samples. A ranking model *f(x)* is then learned from the training data that can assign scores to the feature vectors (documents) and rank the feature vectors using the scores, such that feature vectors with higher degrees are ranked higher. In fact, in the learning process, it takes ranked lists of documents as samples and trains a ranking function by minimizing a listwise loss function based on the predicted and the ground truth lists.

In the field of information retrieval systems, four basic components are formally defined to describe listwise methods: input, output, hypothesis, and loss function [3].

- The input space of the listwise approach contains a set of documents related to the query q, e.g., $x = \left\{x_j\right\}_{j=1}^{m}$.
- The output space of the listwise approach consists of a ranked list (or permutation) of documents. Different types of judgments can be converted into ground truth labels in terms of ranked lists:
  - If the judgment is given as the degree of relevance l_j, all permutations that match the judgment are ground truth permutations. A permutation π_y is defined as consistent with the relevance degree l_j, if $\forall u$, v satisfying $l_u > l_v$, then $\pi_y(u) < \pi_y(v)$ is always true. There might be multiple ground truth permutations in this case. Ω_y is used to denote the set of all such permutations.

  - If the judgment is given as pairwise preferences, then all permutations consistent with the pairwise preferences are ground truth permutations. If the permutation π_y is defined according to the degree of relevance $l_{u,\ v}$ and if ∀u,v $l_{u,\ v} = +1$, then $\pi_y(u) < \pi_y(v)$ is always true. There might be multiple ground truth permutations in this case. Ω_y is used to denote the set of all such permutations.
  - If the judgment is given as the total order of π_l, one can simply define $\pi_y = \pi_l$.

- The hypothesis space contains multivariate functions h that act on a set of documents and predict their permutations. Hypothesis h is usually implemented with a scoring function $h(x) = sort \circ f(x)$. In fact, first, the scoring function f is used to score each document, and then these documents are sorted in descending order of scores to create the desired permutation.
- There are two types of loss functions for listwise approaches. For the first type, the loss function is explicitly related to the evaluation measure (measure-specific loss function). In the second type, there is no loss function (non-specific loss function). Sometimes, it is not easy to determine whether a loss function is listwise or not because some basic components of a listwise loss function may also be pointwise or pairwise. In [3], three distinct criteria are considered to distinguish listwise methods from pointwise and pairwise methods:

  - A loss function is defined in the listwise method according to all training documents related to the query.
  - A loss function in the listwise method cannot be completely decomposed into pairs of documents or separate documents.
  - A loss function in the listwise method emphasizes the concept of a ranked list, and the position of the documents is visible because of the final ranking for the loss function.

According to the loss functions used in the approaches, they can be divided into two subcategories. For the first category, the loss function is explicitly related to evaluation measures. It may be the easiest choice to learn the ranking model by directly optimizing what is used to evaluate the ranking performance. This is precisely the motivation for the first subcategory of listwise approaches, i.e., direct optimization methods. Due to the strong relationship between loss functions and evaluation measures, these algorithms are also referred to as direct optimization methods. The difficulty of optimizing such functions stems from the fact that most existing optimization techniques were developed to deal with continuous and differentiable issues. To solve this limitation,

1. A continuous and differentiable approximation of the measure-based ranking loss can be optimized.
2. One can alternately optimize a continuous and differentiable (and sometimes even convex) bound on the measure-based ranking loss.
3. It is possible to use optimization technologies that can optimize non-smooth objectives.

Table 6.4 Components of listwise approaches

	Listwise methods
Input	Set of document $x = \{x_j\}_{j=1}^{m}$
Output	Ranked list π_y
Hypothesis	Ranking function $f\left(\vec{X}\right)$ sort $\circ$ $f(x)$
Loss function	*Listwise loss function* $L(f; x, \pi_y)$

In the second category of listwise approaches, the loss function is not explicitly related to evaluation measures. In these methods, the loss function, which is not based on a specific measure, reflects the inconsistency between the output of the ranking model and the ground truth permutation.

Table 6.4 shows the components of listwise approaches, separated by input, output, hypothesis, and related loss function.

Listwise methods in recommender systems use the entire list of items viewed by users to optimize a listwise ranking loss function or to optimize the probability of permutations that map items to ranks. Typically, such methods optimize a smooth estimate of a loss function that measures the distance between the reference lists of ranked items in the training data and the ranked list of items generated by the ranking mode.

Next, in Sect. 6.3.3.1, the listwise methods in the field of information retrieval and, in Sect. 6.3.3.2, the listwise methods in the field of recommender systems are discussed and reviewed.

6.3.3.1 Listwise Methods in Information Retrieval

One of the listwise methods based on measure-based loss function minimization that uses measure approximation has been proposed by Qin et al. [43]. The underlying reason for the non-smoothness of the evaluation measure is that the rank positions are non-smooth compared to the ranking scores. Therefore, they suggest making approximations for ranking positions using smooth functions of ranking scores so that the approximate evaluation measure can be derived and optimized. If the summation index in the definition of nDCG is changed from existing positions as a result of ranking to document indexes, nDCG can be defined as Eq. (6.19):

$$Z_m^{-1} \sum_j \left(\frac{G\left(y_j\right)}{\log\left(1 + \pi\left(x_j\right)\right)} \right. \tag{6.19}$$

$\pi(x_j)$ represents the position of document x_j in the ranking list π, which is calculated based on Eq. (6.20):

$$\pi(x_j) = 1 + \sum_{u \neq j} I_{\{f(x_j) - f(x_u) < 0\}} \tag{6.20}$$

From the above equation, it is clear why the nDCG is non-smooth! In fact, nDCG is a smooth function of the rank position; however, the rank position is a non-smooth function of the ranking scores due to the indicator function. The key idea of this method is to approximate the indicator function with a sigmoid function so that the position can be approximated with a smooth function of the ranking points:

$$\widehat{\pi}(x_j) = 1 + \sum_{u \neq j} \left(\frac{\exp\left(-\alpha\left(f(x_j) - f(x_u)\right)\right)}{1 + \exp\left(-\alpha\left(f(x_j) - f(x_u)\right)\right)} \right. \tag{6.21}$$

By replacing $\pi(x_j)$ in Eq. (6.19) with $\widehat{\pi}(x_j)$, we can obtain an approximation for nDCG which is defined by AppNDCG and then define the loss function as (1 − AppNDCG) according to Eq. (6.22).

$$L(f; x, y) = 1 - Z_m^{-1} \sum_{j=1}^{m} \frac{G(y_j)}{\log\left(1 + \widehat{\pi}(x_j)\right)} \tag{6.22}$$

Xu et al. have proposed a method called AdaRank in which the boosting algorithm is used to optimize the exponential function of the evaluation measure [45]. Since the exponential function is monotonic, optimizing the objective function in AdaRank is equivalent to optimizing the evaluation measure itself. In the conventional AdaBoost algorithm, the exponential loss function is used to update the input distribution and determine the combined coefficient of weak learners in each round of iteration. In AdaRank, evaluation measures are used to update the distribution of queries and calculate the combined coefficient of weak rankers. Due to the similarity of this method with AdaBoost, AdaRank can focus on those hard queries and gradually minimize $1-E(\pi, y)$. $E(\pi, y)$ represents the evaluation measure.

Assume that $S = \{(x_i, y_i)\}_{i=1}^{m}$ and the training data are presented as lists of feature vectors and labels (grade). A ranking model $f(x)$ should be learned on the feature vector x. Given a new list of objects (feature vectors) x, the learned ranking model can assign a score to each object. The objects can then be sorted by the scores to create a π (permutation) ranked list. The evaluation is performed at the list level; specifically, a listwise evaluation measure is used as $E(\pi, y)$. In training, a ranking model is developed that can maximize the accuracy in terms of the listwise evaluation measure in the training data or, equivalently, minimize the loss function defined in Eq. (6.23).

$$L(f) = \sum_{i=1}^{m} (1 - E(\pi_i, y_i)) \tag{6.23}$$

where π_i is the permutation of the feature vector x_i with ranking model f and y_i is the corresponding list of grades. This loss function is not smooth and differentiable, so simple evaluation optimization may not work. Instead, since $\exp(-x) \geq 1 - x$ holds, we can consider the optimization of an upper bound of the loss function as Eq. (6.24).

$$\sum_{i=1}^{m} \exp(-E(\pi_i, y_i)) \tag{6.24}$$

Exponential function and logistic function may be used as "surrogate" loss functions in learning. Note that both functions are continuous, differentiable, and even convex with respect to E. One of the advantages of AdaRank is its simplicity, and it is perhaps one of the easiest ways to learn ranking algorithms.

In [46], Cao et al. proposed a listwise method called ListNet, whose loss function is defined using probability distribution on permutations. Many famous models have been proposed to represent permutation probability distributions, such as the Plackett-Luce model or the Mallows model. Since a permutation has a natural one-to-one correspondence with a ranked list, these methods can be applied to ranking. ListNet shows exactly how the Plackett-Luce model can be used to learn rankings. With the Plackett-Luce model, for a given query q, ListNet first defines a permutation probability distribution based on the scores given by the scoring function f. Then it defines the probability distribution of another permutation $P_y(\pi)$ based on the ground truth label. When the ground truth is assumed to be a permutation π_y, $P_y(\pi)$ can be defined as a delta function that takes the value 1 for this permutation only and 0 for all other permutations. It is also possible to first use a mapping function to map the ground truth permutation positions to real-valued scores and then use Eq. (6.25) to calculate the probability distribution. The Plackett-Luce model defines a probability for each possible permutation π of documents, based on the chain rule, as follows:

$$P(\pi|s) = \prod_{j=1}^{m} \frac{\varphi\left(s_{\pi^{-1}(j)}\right)}{\sum_{u=1}^{m} \varphi\left(s_{\pi^{-1}(u)}\right)} \tag{6.25}$$

where $\pi^{-1}(j)$ represents a document in the jth position of permutation of π and φ is a transformation function that can be linear, exponential, or sigmoid. For the next step, ListNet uses the K-L divergence between the probability distribution for the ranking model and for the ground truth to define its loss function. The K-L divergence loss function is convex, and a simple gradient descent method can be used for its optimization. This function is shown in Eq. (6.26):

$$L\left(f;x,\Omega_y\right)=D\left(P_y(\pi)\middle\|P(\pi|(f(w,x)))\right) \tag{6.26}$$

Although the complexity of ListNet testing can be the same as pointwise and pairwise approaches due to the use of a scoring function to define the hypothesis, the complexity of ListNet training is much higher. The complexity of training is an exponential order of m, because K-L divergence loss for each query q requires the addition of m-factorial terms. To deal with this problem, a top k version of K-L divergence loss is introduced in [51], which is based on Plackett-Luce's top k model and can reduce the complexity from exponential to polynomial order.

Map search can usually be divided into two sub-domains, both of which deal with the retrieval and ranking of geospatial entities: The first sub-domain is geocoding, in which, given a map request, it is the task of finding a distinct spatial entity (e.g., a place, a road, or an address) that best matches the query. The second sub-domain is business search, where, given the map query, it finds a ranked list of multiple business entities that best match the query. Zhang et al. have shown how a powerful mechanism, such as attention, can be adapted to design advanced learning to rank (LtR) models for map search [81]. It has been shown that traditional LtR models based on gradient-boosting decision trees (GBDT) and re-ranking provide the retrieved results in map search with relatively high accuracy. In this chapter, a fast attention-based learning to rank learning model is proposed, which uses self-attention with the ranking model.

This model implements very lightweight two-layer attention, which is more compatible to the ranking problem. The first attention layer modifies transformer's self-attention layer in such a way that it can calculate the function *Attention*(Q, K, V) where attention *queries (Q)* are made only on query terms, while attention *keys (K)* and *values (V)* are calculated on the result. The second attention layer processes the features of all heterogeneous inputs and implements a listwise inference function, which shows how users understand the quality of a result in the context of other potential results.

6.3.3.2 Listwise Methods in Recommender Systems

BPR and TOP1 were introduced in the pairwise methods in Sect. 6.3.2.2, which were widely used in recommender systems, but they faced a challenge. The gradient related to the score of a negative sample is the pairwise loss gradient between the target and the sample divided by the number of negative samples. This means that if all negative samples are relevant, their updates still decrease as the number of negative samples increases. To overcome the vanishing of gradients with increasing number of samples, a new type of listwise loss functions called TOP1-max and BPR-max has been proposed by Hidasi et al. in [52], whose idea is that the target score is compared with the most relevant sample score, which is the maximum score among the samples. The choice of max is not differentiable and cannot be used with gradient descent. So, softmax scores are used to preserve differentiability, and

softmax transformations are used only in negative examples. The BPR-max function combines the benefits of pairwise loss, softmax transformation, and score regularization and is calculated according to Eq. (6.27):

$$L_{\mathrm{BPR-max}} = -\log \sum_{j=1}^{N_s} s_j \sigma\left(r_i - r_j\right) + \lambda \sum_{j=1}^{N_s} s_j {r_j}^2 \tag{6.27}$$

BPR-max gradient is the weighted average of individual BPR gradients, where the weights are $s_j\sigma(r_i - r_j)$. The regularization term is performed with softmax with a weight of $\ell 2$ regularization to the scores of the negative samples. λ is the regularization hyperparameter of the loss function.

The TOP1-max loss function is relatively simple. It is not necessary for the regularization section to be applied only to the maximum negative score, but since this mode provides the best results, it is kept as such. A continuous approximation to the maximum choice requires summing the individual loss weighted by the softmax scores s_j. The TOP1-max function is in the form of Eq. (6.28):

$$L_{\mathrm{TOP1-max}} = \sum_{j=1}^{N_s} s_j \left(\sigma\left(r_j - r_i\right) + \sigma\left({r_j}^2\right)\right) \tag{6.28}$$

The TOP1-max gradient is the softmax weighted average of the individual pairwise gradients. If r_j is much less than the maximum negative score, the weight will be almost zero, and more weight will be placed on samples with scores close to the maximum. This solves the gradient vanishing problem with more samples since irrelevant samples are simply ignored, while the gradient points toward the gradient of relevant samples. Of course, if all samples are irrelevant, the gradient approaches zero, but that's all right because if the target score is greater than all sample scores, there's nothing to learn.

Although, in practical applications, only the top, e.g., top N items, in a ranked list are of interest, and the lower-ranked rankings in the list are less reliable, most learning to rank methods optimize the overall rankings of the list. This can potentially reduce the ranking quality of top-ranked items and also have high computational complexity. For this purpose, Liang et al. introduced a listwise ranking model for top N recommendations that directly optimizes a weighted top-heavy truncated ranking objective function, wDCG@N [51]. This model improves the quality of top N item lists by reducing the influence of lower-ranked items and can handle different types of implicit feedback. To solve the limitations of DCG, wDCG@N is defined in [51] as follows:

$$w\mathrm{DCG}_u@N = \sum_{i \epsilon I} 1\left(R_{ui} < N\right).\frac{w_{p_{ui}}\left(2^{y_{ui}} - 1\right)}{\log\left(R_{ui} + 2\right)} \tag{6.29}$$

In Eq. (6.29), $1(R_{ui} < N)$ is the indicator function that selects N top-rated items and ignores other items. w_{pui} is the weight of the implicit feedback p_{ui}, which models the importance and reliability of the feedback. The ranking objective function according to this measure is defined in Eq. (6.30):

$$L(\theta) = \max_{\theta} \sum_{u \epsilon U} \varphi \text{DCG}_u @N - \lambda \|\theta\|_2^2 \tag{6.30}$$

$\|.\|_2^2$ represents L2-norm, and λ is the regularization coefficient. Training a non-smooth objective function like (6.30) is challenging. Hence, it can be replaced by its smooth approximation. The one-sided nature of ReLU removes the contribution of lower-ranked items in the objective function. And besides being computationally simpler, ReLU allows an algorithm with linear computational complexity in the average number of observed items over all users. The objective function of Eq. (6.30), which has been transformed into a smooth objective function by applying the ReLU function, is shown in Eq. (6.31):

$$L^{+}(\theta) = \min_{\theta} - \sum_{u \epsilon U} \sum_{i \epsilon I_u^{+}} h\left(N - R_{ui}^{+}\right). \frac{w_{p_{ui}}(2^{y_{ui}} - 1)}{\log(R_{ui}^{+} + 2)} + \lambda \|\theta\|_2^2 \tag{6.31}$$

Ifada et al. have presented a listwise ranking method for tag-based item recommender systems [53]. This method proposes a new LtR method called Do-Rank by optimizing the DCG for the recommender system as a ranking measure. This method creates an optimal list of recommended items from a DCG perspective for all users. Do-Rank is designed based on a listwise ranking model, which means that the objective function should be formed based on a list of items from each user-tag set. In Do-Rank, $r_{u,i,t}$ represents the ranking position of item i for user u with tag t. To approximate the ranking position of $r_{u,i,t}$ by a smooth function according to the parameters of the model, the smooth function is used, as shown by Eq. (6.32):

$$r_{u,i,t} = 1 + \sum_{j \neq i} \sigma\left(\Delta \widehat{y}\right) \tag{6.32}$$

The function σ is the logistic function, and $\Delta \widehat{y} = \widehat{y}_{u,i,t} - \widehat{y}_{u,j,t}$ is the difference of the predicted relevance scores for two items, which is calculated from the decomposed tensor model. However, in order to implement the listwise model, it is suggested to use the UTS model. The UTS scheme, based on $(u, t) \in A$, interprets observed and non-observed tagging data and labels of each item as positive, null, or negative. Positive input items are derived from the observed data, and negative input items are derived from items not labeled by user u. Using this scheme, the input set consists of a list of label assignments A labeled with their relevance score $y_{u,i,t}$ and uses the following rules:

$$y_{u,i,t} := \begin{cases} 1 & \text{if } (u,i,t) \in A \\ -1 & \text{if} (u,i,t) \notin A \text{ and } i \in I \setminus \{i | (u,i,*) \in A\} \\ 0 & \text{otherwise} \end{cases} \tag{6.33}$$

If $ZP = \{i | y_{u,i,t} = 1\}$ are positive items derived from observed data and $ZN = \{i | y_{u,i,t} = -1\}$ are negative items derived from unlabeled items by user u, the objective function is according to Eq. (6.34):

$$L(\theta) := \sum_{u \epsilon U} \sum_{t \epsilon T} \sum_{i \epsilon ZP} \left(\frac{2^{y_{u,i,t}} - 1}{1 + \log_2 \left(\sum_{j \epsilon ZN} \sigma\left(\Delta \hat{y}\right) \right)} \right) - \lambda_\theta \|\theta\|_F^2 \tag{6.34}$$

The gradient descent method is used to optimize this objective function.

Product rating based on online product reviews is the task of inferring relative user preferences between different products as a type of entity-level sentiment analysis [82]. Product ranking methods usually consist of two parts: (1) understanding the opinions provided by individual reviews and (2) ranking products based on overall user preference, which is the weighted sum of user preferences corresponding to a given category. However, there are some limitations. First, the sentiments in the comments are simplistically categorized as purely positive or negative without considering the various sentiments that lie along the spectrum between these poles. Second, the importance of individual reviews is determined manually by users or independently through product ratings. This importance significantly affects how individual reviews are converted into product ratings. Therefore, there is a need for an integrated approach that extracts distinct information from the observed data and puts it into a ranking score using a module. To this end, Lee et al. have proposed an integrated approach to learn product ratings based on online product reviews [54]. The LtR technique is used to combine many related features into a ranking model. To implement this approach, a hierarchical attention network (HAN), which is a kind of deep neural network, is extended to operate in the domain of ranking with learning strategies. Hierarchical attention network for learning to rank (HAN-LTR) consists of an embedding lookup, a word-level attention-based encoder, a review-level attention-based encoder, a linear layer, and the ranking loss function. Two ranking functions, RankNet and ListNet, have been used to build the ranking model. The architecture of HAN-LTR is shown in Fig. 6.7.

6.3.4 Hybrid Methods

Although listwise methods have been shown to perform better in terms of accuracy than pointwise and pairwise approaches [46], the need to improve the performance of LtR approaches has prompted researchers to propose hybrid methods. For example, Sculley proposed an LtR approach that simultaneously combines pointwise

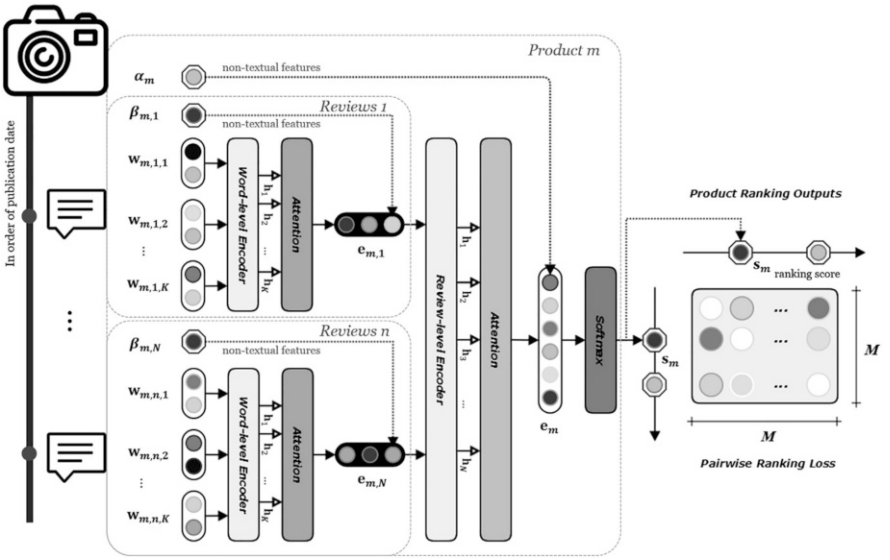

Fig. 6.7 The architecture of HAN-LTR [54]

using linear regression and pairwise with a support vector machine [55]. Two other hybrid approaches are LambdaRank and LambdaMART, which combine pairwise with listwise methods [56]. LambdaRank is based on RankNet [26], while LambdaMART is a boosted LambdaRank tree. Both LambdaMART and LambdaRank have shown good performance in terms of data retrieval accuracy.

In this section, some important research using hybrid LtR approaches are discussed and reviewed.

Sadek et al. have presented an effective and efficient method for LtR in the field of IR, which combines evolutionary strategy (ES) with machine learning techniques [57]. The proposed method is an ES that creates a vector of weights, each representing a desirable document feature. Three methods for initializing the weight vector (chromosome) are addressed in this chapter: ES-Rank simply sets all the genes of the initial chromosome to the same value. IESR-Rank uses linear regression, and IESVM-Rank uses a support vector machine for the initialization process.

Pointwise and pairwise approaches have their complementary advantages and disadvantages, and many studies focus on only one approach and try to reduce their drawbacks in case-by-case methods. Cinar et al. extended a hybrid pointwise-pairwise standard paradigm for LtR in the context of personalized recommendations [58]. For this purpose, a new surrogate loss is introduced, which is an optimal and adaptive combination of these two approaches, so that the exact balance between pointwise and pairwise contributions can depend on the particular pair or triplet instance.

In this method, a learning strategy is proposed in which the model itself can make the correct decision: which pointwise or pairwise approaches should be adopted for

each triple $<u, i, j>$? The surrogate loss presented in this chapter considers that there is a continuum between pointwise and pairwise approaches, and how to position the cursor in this continuum should be learned from the data and depend on the triplet considered. This can be shown using the coefficient γ, which softly determines the trade-off between pointwise and pairwise rank:

$$p(i > j|u) = \sigma(f(u, i|\theta) - \gamma f(u, j|\theta)) \tag{6.35}$$

In Eq. (6.35), instead of γ taking only the values [0,1] and being a simple hyperparameter, it is proposed to be calculated as a learnable function that depends on the user u and items i and j and called "adaptive mixing function."

The method presented by D'Amico et al. focuses on the problem of session-based and context-aware accommodation recommendations in the travel domain [59]. The purpose of this chapter is to recommend suitable accommodation according to the traveler's needs to maximize the chance of changing the direction (click out) to the booking site, relying on explicit and implicit signals of the user in one session (clicks, search modification, filter use). For this purpose, they used a session's contextual and content features. Contextual features exploit interactions with accommodations occurring within a session. For content features, interactions between sessions or other non-session-related information are considered. The proposed model relies on gradient boosting for decision trees and combines different methods. In order to exploit the sequential structure of the problem, a recurrent neural network has been developed in which each session is represented with a fixed number of interactions and fed to the network. This network uses TensorFlow ranking, which is a ranking algorithm published by Google and is able to optimize listwise losses for ranking.

Although many MF-based methods have performed well in recommender systems, they cannot effectively learn user and item representations, making them poorly capable of capturing complex and deeper information about the interaction between users and items. To solve this problem, and inspired by the great success of deep learning methods applied to LtR, DeepRank was proposed by Chen et al. [2]. Instead of predicting rank, DeepRank uses predicted scores. In fact, the proposed ranking model is reduced from top N listwise to top one listwise, which is a simpler structure, and then to the simplest possible structure, which is the pairwise learning method, which is one of the most popular learning to rank methods in recommender systems.

Considering the user-item rating matrix R, where n is the number of users and m is the number of items, the interaction matrix Y is defined as Eq. (6.36):

$$y_{ui} = \begin{cases} 1, & \text{if } r_{ui} > 0 \\ 0, & \text{else} \end{cases} \tag{6.36}$$

where $y_{ui} \in Y$ and $r_{ui} \in R$ indicate the rating of user u to item i. The main purpose of DeepRank is to predict the unrated order of items based on their final score. For this purpose, the objective function is defined by Eq. (6.37):

$$L = f(y, \hat{y}) + \lambda \Omega(\Theta) \tag{6.37}$$

In the above equation, *f(.)* is the loss function of the model, y and $\hat{y}$ are the correct and predicted labels of the samples, and $\Omega(\Theta)$ is the regularization function to reduce overfitting. Due to recent advances in adversarial learning, there is a strong and continuous interest in investigating how adversarial LtR is performed. Despite the successes of adversarial methods for LtR, there are still many open issues in this field. For example, previous methods have focused on optimizing pointwise or pairwise learning to rank functions or have only investigated how to adapt the GAN framework for ranking. While GAN has various types, in this regard, Yu et al. have done an in-depth study of how to perform adversarial learning to LtR by adapting different adversarial learning frameworks [60].

6.4 Ranking Aggregation

Ranking aggregation is a problem that has been widely studied in various domains, such as meta-search and image integration. The main purpose of ranking aggregation is to create an overall ranking from the ranking results of items or alternatives, which uses multiple ranking functions to find better functions. These methods can be classified into two categories: score-based methods and order-based methods.

In score-based methods, ranked lists of items are scored separately, and these scores are used by the ranking aggregation function [61, 62]. On the other hand, order-based or rank-based methods only use the order of items in separate ranked lists. These methods are widely used in modern meta-search engines because of their simplicity and linear time complexity with the number and size of ranked lists. In some situations, it is difficult to find ranking points; hence, order-based methods seem to be the best choice because they only use the relative position of the items in each ranked list and are called position-based methods [34].

Ranking aggregation methods are classified into two categories based on another approach [86]: supervised and unsupervised learning methods. Supervised learning-based methods use a training set for ranking. But in methods based on unsupervised learning, ranking aggregation is created based on distance measures and provides the possibility of comparing individual rankings with them.

An important field in ranking aggregation is recommender systems. Among recommender systems, top N recommenders work by recommending the ranking of N items that can be interesting to the user. They often differ in the rankings of the items they return to the user, and they provide an opportunity to improve the final recommendation ranking by aggregating the outputs of different algorithms. The advantages of using ranking aggregation methods in recommender systems are [83]:

1. Providing more accurate item recommendations to users, taking into account the different biases of the recommenders
2. Improving the diversity of recommendation rankings

3. Reducing the impact of items imprecisely placed in high positions by a recommender

In the following subsections, ranking aggregation methods in the scope of information retrieval and recommender systems will be discussed.

6.4.1 Ranking Aggregation Methods in Information Retrieval

One of the first approaches in ranking aggregation is the Borda count, which is based on the unsupervised method for ranking aggregation suggested by Aslam and Montague in the meta-search [61]. The Borda count ranks a fixed set of c candidates in the order of preference for each voter. For each voter, the top-ranked candidate is given c points, the second-ranked candidate is given $c-1$ points, and so on. If a number of candidates are left unranked by the voter, the remaining points are divided equally among the unranked candidates. Candidates are ranked in the order of total points, and the candidate who gets the most points wins the election.

The Borda count finalizes the ranking of documents based on their position in the base ranking. More precisely, in the final ranking, the documents are sorted according to the number of documents that rank below them. If a document ranks high in many base rankings, it will also rank high in the final ranking list.

Unsupervised ranking aggregation methods use majority voting in their final ranking decisions. In fact, the methods treat all primary ranking lists equally and assign high scores to documents that rank high in most primary ranking lists. For example, in meta-search, ranking lists produced by different search engines may have different accuracy and reliability. One might want to learn the weights of the primary ranking lists. Supervised learning methods such as cranking proposed by Lebanon and Lafferty can solve the problem [63]. This method uses the probabilistic model of Eq. (6.38):

$$P\left(\pi|\theta, \sum\right) = \frac{1}{Z(\theta, \sum)} \exp\left(\sum_{j=1}^{k} \theta_j .\ d\left(\pi, \sigma_j\right)\right) \tag{6.38}$$

In Eq. (6.38), π is the final ranking, $\sum = (\sigma_1, \ldots, \sigma_k)$ is the basic ranking, d is the distance between two rankings, and θ is the weight parameters. For example, d can be calculated based on Kendal's tau. Maximum likelihood estimation can be used to learn model parameters. If the final ranking and base rankings are all complete ranking lists in the training data, the log-likelihood function is calculated as Eq. (6.39):

$$L(\theta) = \log \prod_{i=1}^{m} P\left(\pi_i | \theta, \sum_i\right) = \sum_{i=1}^{m} \log \frac{\exp\left(\sum_{j=1}^{k} \theta_j . d\left(\pi_i, \sigma_{i,j}\right)\right)}{\sum_{\pi_i \epsilon \Pi} exp \sum_{j=1}^{k} \theta_j . d\left(\pi_i, \sigma_{i,j}\right)} \tag{6.39}$$

In the above relation, π_i represents partial lists.

Liang et al. focused on combining ranked lists of documents that are retrieved in response to a query, and based on this, they proposed multiple learning aggregation approaches, ManX and v-ManX, which are based on the cluster hypothesis and exploit inter-document similarity information [65]. ManX is a new manifold-based data fusion approach that (1) is based on the generic data fusion method X and (2) allows similar documents to support each other by using inter-document similarities in a global manifold of fused documents. To further improve rank aggregation performance, a virtual adversarial manifold learning algorithm, v-ManX, and an efficient version that uses anchor documents, a-v-ManX, are proposed. The proposed virtual adversarial multiple learning algorithms first create a virtual adversarial document for each original document and then regularize the model to produce the same output distribution model according to the document they produce in the adversarial perturbation.

Instance search is a less studied subject in the information retrieval domain, described as a search in which a new set of results is returned for each keystroke. If the goal is to provide a wide range of results in a single list beyond lexical matches, implementing a robust instant search service presents several challenges, including combining results for a given search term when there are multiple matches (lexical, semantic, etc.) or preventing ancillary matches from ranking higher than logical matches when there are multiple candidate sources. In this regard, Rome et al. have proposed a solution to solve these challenges in a real platform (Xfinity) that has millions of users [67]. The presented method consists of three stages: candidate generation, availability filtering, and re-ranking of key components. In the candidate generation stage, asynchronous calls are established with multiple indices. Then, the candidates are filtered based on the item's availability to the user. In the next step, the results are combined into a single list through a heuristic method and then sent to the re-ranking stage. The re-ranking stage consists of two deep learning models combining different candidate lists and fine-tuning the user results. In the final stage of the pipeline, business logic is applied to create the final ranking and to respect the product requirements.

Motivated by the fact that current ranking aggregation methods are feature-unaware or sensitive to noisy features, Chiang et al. proposed a new rank aggregation model that learns rank scores from features and comparisons simultaneously [68]. In this method, the ranking is estimated by balancing between pairwise comparisons and feature information (Rank Aggregation by Balancing Feature, RABF). One of the highlights of this model is its improved sample complexity guarantee. Sample complexity analysis for ranking aggregation has recently

attracted more attention intending to study the number of comparisons required to ensure ranking accuracy.

To reduce the computational complexity of global ranking learning, a common method is to use rank breaking. In rank breaking, the collected ordinal data are first transformed into a bag of pairwise comparisons, ignoring the dependencies in the original data. It is then processed through existing inference algorithms designed for independent pairwise comparisons, hoping that the dependence on the input data does not cause bias. This idea is one of the main motivations in several approaches that are used in learning to rank from pairwise comparisons. However, it has been shown that such a heuristic of full rank breaking, where all pairs are weighted and treated equally, leads to estimation bias. The key idea to produce accurate and unbiased estimates is to treat pairwise comparisons unequally, depending on the topology of the collected data. For this purpose, Khetan et al. have investigated how the accuracy depends on the topology of the data and the weights of pairwise comparisons [66].

6.4.2 Ranking Aggregation Methods in Recommender Systems

A number of ranking aggregation methods have been presented in the field of recommender systems. Tulio et al. presented a hybrid recommendation algorithm that aggregates the results of different input recommendation algorithms to improve the accuracy, novelty, and diversity of rankings [69]. Aggregation is performed using a weighted linear combination of the items returned by the recommendation algorithms. The weights are optimized by an evolutionary algorithm following a Pareto-efficient multi-objective setting that considers the accuracy, novelty, and diversity of ranking aggregation. Indeed, the overemphasis on the accuracy of the recommendations can cause information over-specialization and make recommendations boring and even predictable. Novelty and diversity are two useful solutions to these problems, which have recently been studied by Jafari and Ravanmehr [84, 85].

Following evolutionary approaches, papers [70] and [71] proposed rank aggregation methods to combine rankings generated by top N recommendation algorithms. Oliveira et al. proposed ERA (evolutionary rank aggregation), which uses genetic programming (GP) to evolve a population of aggregation functions (individuals) [70]. These functions are used to assign scores to the items in the input ranking, and then by sorting the items according to their scores, the ranking is aggregated.

Silva et al. used a genetic algorithm (GA) to combine different rankings from memory-based collaborative filtering [71]. The strategy behind the proposed GA consists of building a structure for each user that is used to select the items that form the ranking aggregation. The proposed GA establishes a structure that determines

how many items should be selected from each input ranking to aggregate the ranking.

One of the possible approaches for ranking aggregation is to aggregate the ranking to produce a single-criteria matrix and thus create a ranking list. This approach hides important details in the ranking of each criterion. Abderrahmane et al. proposed a three-stage hybrid ranking order for multi-criteria recommender systems that uses rank aggregation [72]. This system generates a partial ranking list using learning to rank and then generates a final ranked list using rank aggregation. This method is considered a movie recommender system, and for each movie, the criteria of actor, director, story, and visual are used, and it uses many useful ranking methods to reach the final ranked list. In the first step of this method (matrix decomposition), the multi-criteria user-item matrix is decomposed into N single-rating user-item matrix according to the number of criteria in the system. Hence, five single-rating user-item matrices are obtained. In the second step (learning to rank), one of the LtR methods is applied in each matrix separately to find partially ranked lists. For this purpose, a matrix factorization based listwise method is used to sort the items for each matrix separately and minimizes an error function that represents the uncertainty between the input training list and the resulting output list. In the third stage (rank aggregation), it ranks the partially ranked lists using rank aggregation. The proposed framework of [72] is shown in Fig. 6.8.

Zhao et al. consider video recommendation as a ranking problem and create multiple ranking lists by exploring different information sources [73]. For this purpose, a multi-task rank aggregation approach is proposed to integrate the ranking lists for different users in a joint mode. The proposed approach organizes related data according to information sources about users, including profiles, viewing history, and information available on social networks, as well as video information. After creating several video ranking lists using different data sources, the next task is to aggregate these video lists into an optimized video list so that the best ones can be recommended to users. A score-based approach is used here, i.e., the ranking lists are combined according to the ranking scores of each video instead of their ranking positions.

6.5 Discussion

Learning to rank methods are based on machine learning techniques in ranking results in different domains, most notably in information retrieval and recommender systems. These methods are classified into three general categories: pointwise, pairwise, and listwise.

Pointwise methods are usually defined based on classification [9–12], regression [7, 8], or ordinal regression [13–15, 17]. Pointwise methods are also used in the field of recommender systems to rank the results [19, 20]. Instead of focusing on the personalized ranking of a set of items, pointwise methods only focus on predicting the exact ranking of an item. Users tend to pay more attention to an item's ranking

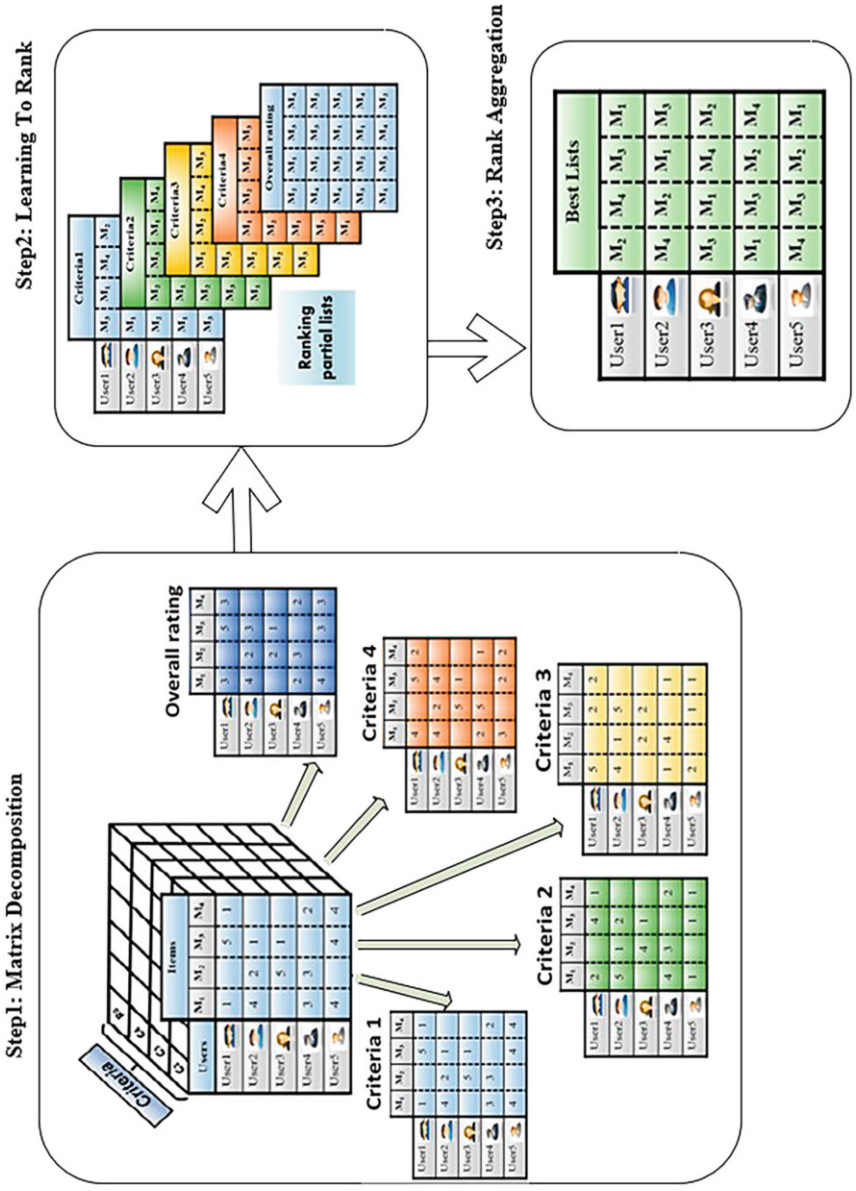

Fig. 6.8 The proposed framework of [72]

order than its rating. For example, when a user wants to watch a movie online, he often cares less about its rating and chooses the movie at the top of the recommended list. Compared to other methods, pointwise approaches do not consider the dependencies between documents; therefore, the position of a document in the final ranking list is not visible to their loss function. Therefore, the pointwise loss function may overemphasize unimportant documents (documents that rank low in the final ranking list and therefore do not affect user experiences) [3].

In fact, the advantages of pointwise ranking are twofold. First, the pointwise ranking is calculated separately based on each query-document pair (q), which makes it simple and easy to scale. Second, the outputs of the neural models learned with the pointwise loss function often have real values in practice. For example, in sponsored search, a model learned with cross-entropy loss and click-through rate can directly predict the probability of a user clicking on search ads, which is more important than creating a good result list in some application scenarios. However, in general, the use of pointwise ranking in ranking operations is actually less effective. Since pointwise loss functions do not consider any document preference or ordered information, they are not guaranteed to produce the best-ranked list with minimum model loss. Therefore, effective ranking paradigms that directly optimize document ranking based on pairwise and listwise loss functions have often been proposed for LtR problems.

The pairwise approach does not focus on accurately predicting the degree of relevance of each document, but rather the relative order between two documents, and is closer to the concept of "ranking" than the pointwise approach. In the pairwise approach, ranking is usually reduced to classification over pairs of documents, determining which document is preferred in a pair. This classification differs from the classification in the pointwise approach because it operates on both documents under review. The proposed pairwise ranking learning approaches have different types, some of which utilize neural networks [34], perceptrons [31], boosting [33], SVM [29], and other machine learning methods [28, 30]. It is worth mentioning that pairwise methods are widely used in the field of session-based recommender system [36–38].

However, pairwise methods have two major problems: (1) They only consider the relative order of two items, not the position of the items in the proposed list. As a result, items at the top of the suggested list are more important than items at the bottom. If the items at the top are evaluated incorrectly, the ranking cost is significantly higher than the items at the bottom. (2) The number of relevant items varies widely among different users. After converting to item pairs, some users have hundreds of corresponding item pairs. While others have only tens of pairs of corresponding items, it is difficult to evaluate the performance of the models.

Ideally, when pairwise ranking loss is minimized, all preference relations between documents should be satisfied, and the model will produce the optimal result list for each query. This makes pairwise ranking objectives effective in many tasks where system performance is evaluated based on the ranking of relevant documents. However, in practice, optimizing document preferences in pairwise methods does not always improve the final ranking measure for two reasons: (1) It is impossible to

develop a ranking model that can correctly predict document preferences in all cases, and (2) not all document pairs are equally important in the computation of most existing ranking measures. This means that the prediction performance of pairwise preference is not equal to the performance of the final retrieval results as a list.

Listwise approaches deal with the ranking problem more naturally than other learning to rank methods. In particular, it considers ranked lists as examples of the learning and prediction process. The group structure of the ranking is preserved, and the ranking evaluation measure can be more directly incorporated into the loss functions. Different listwise methods have been proposed for information retrieval, such as [43, 45–50, 81], and some of them have been utilized in the scope of recommender systems, such as [52–54].

Note that for the listwise approach, the output space that facilitates the learning process is exactly the output space of the problem. In this regard, the theoretical analysis of the listwise approach can have a more direct value for understanding the real ranking problem than other approaches in which there is a mismatch between the output spaces. This feature facilitates learning and the actual output space of the problem.

While listwise ranking objectives are usually more effective than pairwise ranking objectives, their high computational cost often limits their applications. In fact, these methods are suitable for the re-ranking stage in a small set of candidate documents. Since many practical search systems now use neural network models to re-rank documents, listwise ranking objectives have become increasingly popular in neural ranking frameworks.

6.6 Conclusion

Learning to rank (LtR) has emerged in recent years based on a combination of machine learning and information retrieval. Learning to rank uses machine learning methods to rank the results and solve a different problem than the classic recommender problem of predicting ratings. Ranking models and algorithms help recommender systems arrange the items of the list of recommendations in the most optimal state possible. The important challenges in this field are related to ranking creation and ranking aggregation.

Ranking creation is related to the automatic construction of the ranking model using machine learning methods. In this chapter, we mainly introduced four approaches to ranking creation: pointwise, pairwise, listwise, and hybrid. The pointwise approach reduces ranking to regression, classification, or ordinal regression on every single item. The pairwise approach formulates ranking as a pairwise classification problem. The listwise approach, which regards ranking as a new problem, tries to directly optimize the non-smooth IR evaluation measures or to minimize listwise ranking losses. To improve the performance, researchers proposed hybrid LtR approaches.

Ranking aggregation combines multiple rankings into a single ranking and creates an overall ranking from the ranking results of items.

References

1. Hang Li. "Learning to rank for information retrieval and natural language processing." *Synthesis lectures on human language technologies* 7, no. 3 (2014): 1-121. https://doi.org/10.1007/978-3-031-02155-8
2. Ming Chen, and Xiuze Zhou. "DeepRank: Learning to rank with neural networks for recommendation." Knowledge-Based Systems 209 (2020): 106478. https://doi.org/10.1016/j.knosys.2020.106478
3. Tie-Yan Liu. "Learning to Rank for Information Retrieval.", Springer Berlin Heidelberg, 2011. https://doi.org/10.1007/978-3-642-14267-3
4. Chengxiang Zhai, and John Lafferty. "A study of smoothing methods for language models applied to information retrieval." ACM Transactions on Information Systems (TOIS) 22, no. 2 (2004): 179-214. https://doi.org/10.1145/984321.984322
5. Kim Falk. Practical recommender systems. Simon and Schuster, 2019.
6. Meike Zehlike, Ke Yang, and Julia Stoyanovich. "Fairness in ranking, part ii: Learning-to-rank and recommender systems." ACM Computing Surveys 55, no. 6 (2022): 1-41. https://doi.org/10.1145/3533380
7. David Cossock, and Tong Zhang. "Subset ranking using regression." In Learning Theory: 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006. Proceedings 19, pp. 605-619. Springer Berlin Heidelberg, 2006. https://doi.org/10.1007/11776420_44
8. Norbert Fuhr. "Optimum polynomial retrieval functions based on the probability ranking principle." ACM Transactions on Information Systems (TOIS) 7, no. 3 (1989): 183-204. https://doi.org/10.1145/65943.65944
9. Ramesh Nallapati. "Discriminative models for information retrieval." In Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, Sheffield, United Kingdom, July 25-29, 2004, pp. 64-71. https://doi.org/10.1145/1008992.1009006
10. Fredric C. Gey. "Inferring probability of relevance using the method of logistic regression." In Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, July 3-6, 1994, pp. 222-231. https://doi.org/10.1007/978-1-4471-2099-5_23
11. Ping Li, Qiang Wu, and Christopher Burges. "Mcrank: Learning to rank using multiple classification and gradient boosting." Advances in neural information processing systems 20 (2007).
12. Adriano A. Veloso, Humberto M. Almeida, Marcos A. Gonçalves, and Wagner Meira Jr. "Learning to rank at query-time using association rules." In Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, pp. 267-274. 2008. https://doi.org/10.1145/1390334.1390381
13. Koby Crammer, and Yoram Singer. "Pranking with ranking." In Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, pp. 641-647. 2001.
14. Edward F. Harrington. "Online ranking/collaborative filtering using the perceptron algorithm." In Proceedings of the 20th International Conference on Machine Learning (ICML-03), pp. 250-257. 2003.
15. Amnon Shashua, and Anat Levin. "Ranking with large margin principle: Two approaches." Advances in neural information processing systems 15 (2002).

16. Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 368-377. 2018. https://doi.org/10.1145/3219819.3219846
17. Jason DM Rennie, and Nathan Srebro. "Loss functions for preference levels: Regression with discrete ordered labels." In Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling, Palo Alto, California, USA, March 21–23, 2005, vol. 1, AAAI Press, Menlo Park, CA, 2005.
18. Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts." In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, London, United Kingdom, August 19 - 23, 2018, pp. 1930-1939. https://doi.org/10.1145/3219819.3220007
19. Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. "Recommending what video to watch next: a multitask ranking system." In Proceedings of the 13th ACM Conference on Recommender Systems, pp. 43-51. 2019. https://doi.org/10.1145/3298689.3346997
20. Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. "Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations." In Proceedings of the 14th ACM Conference on Recommender Systems, Brazil, September 22-26, 2020, pp. 269-278. https://doi.org/10.1145/3383313.3412236
21. Ayan Sinha, David F. Gleich, and Karthik Ramani. "Deconvolving feedback loops in recommender systems." Advances in neural information processing systems 29 (2016).
22. Nengjun Zhu, Jian Cao, Xinjiang Lu, and Qi Gu. "Leveraging pointwise prediction with learning to rank for top-N recommendation." World Wide Web 24 (2021): 375-396. https://doi.org/10.1007/s11280-020-00846-3
23. Christopher Burges, Robert Ragno, and Quoc Le. "Learning to rank with nonsmooth cost functions." Advances in neural information processing systems 19 (2006).
24. Ralf Herbrich, Obermayer, K., Graepel, T. "Large margin rank boundaries for ordinal regression". In: Advances in Large Margin Classifiers (2000).
25. Thorsten Joachims. "Optimizing search engines using clickthrough data." In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton Alberta, Canada, July 23-26, 2002, pp. 133-142. https://doi.org/10.1145/775047.775067
26. Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. "Learning to rank using gradient descent." In Proceedings of the 22nd international conference on Machine learning, Bonn, Germany, August 7-11, 2005, pp. 89-96. https://doi.org/10.1145/1102351.1102363
27. Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. "High accuracy retrieval with multiple nested ranker." In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, Seattle, USA, August 6-11, 2006, pp. 437-444. https://doi.org/10.1145/1148170.1148246
28. Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. "Frank: a ranking method with fidelity loss." In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, Amsterdam, Netherlands, July 23-27, 2007, pp. 383-390. https://doi.org/10.1145/1277741.1277808
29. Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. "Adapting ranking SVM to document retrieval." In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, Seattle, USA, August 6-11, 2006, pp. 186-193. https://doi.org/10.1145/1148170.1148205
30. Zhaohui Zheng, Hongyuan Zha, Tong Zhang, Olivier Chapelle, Keke Chen, and Gordon Sun. "A general boosting method and its application to learning ranking functions for web search." Advances in neural information processing systems 20 (2007).

31. Libin Shen, and Aravind K. Joshi. "Ranking and reranking with perceptron." Machine Learning 60 (2005): 73-96. https://doi.org/10.1007/s10994-005-0918-9
32. Massih Reza Amini, Tuong Vinh Truong, and Cyril Goutte. "A boosting algorithm for learning bipartite ranking functions with partially labeled data." In Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval, Singapore, July 20 -24, 2008, pp. 99-106. https://doi.org/10.1145/1390334.1390354
33. Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. "An efficient boosting algorithm for combining preferences." Journal of machine learning research 4, no. Nov (2003): 933-969.
34. Leonidas Akritidis, Dimitrios Katsaros, and Panayiotis Bozanis. "Effective rank aggregation for metasearching." Journal of Systems and Software 84, no. 1 (2011): 130-143. https://doi.org/10.1016/j.jss.2010.09.001
35. Wang Zhou, Yujun Yang, Yajun Du, and Amin Ul Haq. "Pairwise deep learning to rank for top-N recommendation." Journal of Intelligent & Fuzzy Systems 40, no. 6 (2021): 10969-10980. https://doi.org/10.3233/JIFS-202092
36. Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. "BPR: Bayesian personalized ranking from implicit feedback." In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal Quebec, Canada, June 18 - 21, 2009, pp. 452-461.
37. Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In Proceedings International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016. https://doi.org/10.48550/arXiv.1511.06939
38. Hongzhi Liu, Zhonghai Wu, and Xing Zhang. "CPLR: Collaborative pairwise learning to rank for personalized recommendation." Knowledge-Based Systems 148 (2018): 31-40. https://doi.org/10.1016/j.knosys.2018.02.023
39. Yulu Du, Xiangwu Meng, Yujie Zhang, and Pengtao Lv. "GERF: A group event recommendation framework based on learning-to-rank." IEEE Transactions on Knowledge and Data Engineering 32, no. 4 (2019): 674-687. https://doi.org/10.1109/TKDE.2019.2893361
40. Murat Yagci, Tevfik Aytekin, and Fikret Gurgen. "On parallelizing SGD for pairwise learning to rank in collaborative filtering recommender systems." In Proceedings of the Eleventh ACM Conference on Recommender Systems, Como, Italy, August 27-31, 2017. pp. 37-41. https://doi.org/10.1145/3109859.3109906
41. Weiwen Liu, Qing Liu, Ruiming Tang, Junyang Chen, Xiuqiang He, and Pheng Ann Heng. "Personalized Re-ranking with Item Relationships for E-commerce." In Proceedings of the 29th ACM International Conference on Information & Knowledge Management, Ireland, October 19-23, 2020, pp. 925-934. https://doi.org/10.1145/3340531.3412332
42. Malte Ludewig, and Dietmar Jannach. "Learning to rank hotels for search and recommendation from session-based interaction logs and meta data." In Proceedings of the Workshop on ACM Recommender Systems Challenge, Copenhagen, Denmark, September 20, 2019, pp. 1-5. https://doi.org/10.1145/3359555.3359561
43. Tao Qin, Tie-Yan Liu, and Hang Li. "A general approximation framework for direct optimization of information retrieval measures." Information retrieval 13 (2010): 375-397. https://doi.org/10.1007/s10791-009-9124-x
44. Michal Rolínek, Vít Musil, Anselm Paulus, Marin Vlastelica, Claudio Michaelis, and Georg Martius. "Optimizing rank-based metrics with blackbox differentiation." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, USA, June 14-19, 2020 pp. 7620-7630.
45. Jun Xu, and Hang Li. "Adarank: a boosting algorithm for information retrieval." In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, Amsterdam, Netherlands, July 23-27, 2007, pp. 391-398. https://doi.org/10.1145/1277741.1277809
46. Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. "Learning to rank: from pairwise approach to listwise approach." In Proceedings of the 24th international conference

on Machine learning, Corvallis, USA, June 20-24, 2007, pp. 129-136. https://doi.org/10.1145/1273496.1273513
47. Robin Swezey, Aditya Grover, Bruno Charron, and Stefano Ermon. "Pirank: Scalable learning to rank via differentiable sorting." Advances in Neural Information Processing Systems 34 (2021): 21644-21654.
48. Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. "Setrank: Learning a permutation-invariant ranking model for information retrieval." In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, China, July 25-30, 2020, pp. 499-508. https://doi.org/10.1145/3397271.3401104
49. Fatih Cakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. "Deep metric learning to rank." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Long Beach, CA, USA, June 16-17, 2019, pp. 1861-1870.
50. Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. "Learning a deep listwise context model for ranking refinement." In The 41st international ACM SIGIR conference on research & development in information retrieval, pp. 135-144. 2018. https://doi.org/10.1145/3209978.3209985
51. Junjie Liang, Jinlong Hu, Shoubin Dong, and Vasant Honavar. "Top-N-rank: A scalable list-wise ranking method for recommender systems." In 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, December 10-13, 2018, pp. 1052-1058. https://doi.org/10.1109/BigData.2018.8621994
52. Balázs Hidasi, and Alexandros Karatzoglou. "Recurrent neural networks with top-k gains for session-based recommendations." In Proceedings of the 27th ACM international conference on information and knowledge management, Torino Italy October 22 - 26, 2018, pp. 843-852. https://doi.org/10.1145/3269206.3271761
53. Noor Ifada, and Richi Nayak. "Do-rank: DCG optimization for learning-to-rank in tag-based item recommendation systems." In Advances in Knowledge Discovery and Data Mining: 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part II 19, pp. 510-521. https://doi.org/10.1007/978-3-319-18032-8_40
54. Ho-Chang Lee, Hae-Chang Rim, and Do-Gil Lee. "Learning to rank products based on online product reviews using a hierarchical deep neural network." Electronic Commerce Research and Applications 36 (2019): 100874. https://doi.org/10.1016/j.elerap.2019.100874
55. David Sculley. "Combined regression and ranking." In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, USA, July 25-28, 2010 pp. 979-988. https://doi.org/10.1145/1835804.1835928
56. Christopher JC. Burges. "From ranknet to lambdarank to lambdamart: An overview." Learning 11, no. 23-581 (2010): 81.
57. Osman Ali Sadek Ibrahim, and Dario Landa-Silva. "An evolutionary strategy with machine learning for learning to rank in information retrieval." Soft Computing 22 (2018): 3171-3185. https://doi.org/10.1007/s00500-017-2988-6
58. Yagmur Gizem Cinar, and Jean-Michel Renders. "Adaptive pointwise-pairwise learning-to-rank for content-based personalized recommendation." In Proceedings of the 14th ACM Conference on Recommender Systems, Brazil, September 22 - 26, 2020, pp. 414-419. https://doi.org/10.1145/3383313.3412229
59. Edoardo D'Amico, Giovanni Gabbolini, Daniele Montesi, Matteo Moreschini, Federico Parroni, Federico Piccinini, Alberto Rossettini, Alessio Russo Introito, Cesare Bernardis, and Maurizio Ferrari Dacrema. "Leveraging laziness, browsing-pattern aware stacked models for sequential accommodation learning to rank." In Proceedings of the Workshop on ACM Recommender Systems Challenge, Copenhagen, Denmark, September 20, 2019, pp. 1-5. https://doi.org/10.1145/3359555.3359563
60. Hai-Tao Yu, Rajesh Piryani, Adam Jatowt, Ryo Inagaki, Hideo Joho, and Kyoung-Sook Kim. "An in-depth study on adversarial learning-to-rank." Information Retrieval Journal 26, no. 1 (2023): 1. https://doi.org/10.1007/s10791-023-09419-0

61. Javed A. Aslam, and Mark Montague. "Models for metasearch." In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans Louisiana, USA, pp. 276-284. 2001. https://doi.org/10.1145/383952.384007
62. Christopher C. Vogt, and Garrison W. Cottrell. "Fusion via a linear combination of scores." Information retrieval 1, no. 3 (1999): 151-173. https://doi.org/10.1023/A:1009980820262
63. Guy Lebanon, and John Lafferty. "Cranking: Combining rankings using conditional probability models on permutations." In ICML, vol. 2, pp. 363-370. 2002.
64. Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. "Rank aggregation methods for the web." In Proceedings of the 10th international conference on World Wide Web, Hong Kong, May 1 - 5, 2001, pp. 613-622. https://doi.org/10.1145/371920.372165
65. Shangsong Liang, Ilya Markov, Zhaochun Ren, and Maarten de Rijke. "Manifold learning for rank aggregation." In Proceedings of the 2018 World Wide Web Conference, Lyon, France, April 23 - 27, 2018, pp. 1735-1744. https://doi.org/10.1145/3178876.3186085
66. Ashish Khetan, and Sewoong Oh. "Data-driven rank breaking for efficient rank aggregation." In International Conference on Machine Learning, pp. 89-98. PMLR, 2016.
67. Scott Rome, Sardar Hamidian, Richard Walsh, Kevin Foley, and Ferhan Ture. "Learning to Rank Instant Search Results with Multiple Indices: A Case Study in Search Aggregation for Entertainment." In Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11-15, 2022, pp. 3412-3416. https://doi.org/10.1145/3477495.3536334
68. Kai-Yang Chiang, Cho-Jui Hsieh, and Inderjit Dhillon. "Rank aggregation and prediction with item features." In Artificial Intelligence and Statistics, pp. 748-756. PMLR, 2017.
69. Marco Tulio Ribeiro, Nivio Ziviani, Edleno Silva De Moura, Itamar Hata, Anisio Lacerda, and Adriano Veloso. "Multiobjective pareto-efficient approaches for recommender systems." ACM Transactions on Intelligent Systems and Technology (TIST) 5, no. 4 (2014): 1-20. https://doi.org/10.1145/2629350
70. Samuel Oliveira, Victor Diniz, Anisio Lacerda, and Gisele L. Pappa. "Evolutionary rank aggregation for recommender systems." In 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, July 24-29, 2016, pp. 255-262. https://doi.org/10.1109/CEC.2016.7743803
71. Edjalma Queiroz da Silva, Celso G. Camilo-Junior, Luiz Mario L. Pascoal, and Thierson C. Rosa. "An evolutionary approach for combining results of recommender systems techniques based on collaborative filtering." Expert Systems with Applications 53 (2016): 204-218. https://doi.org/10.1016/j.eswa.2015.12.050
72. Abderrahmane Kouadria, Omar Nouali, and Mohammad Yahya H. Al-Shamri. "A multi-criteria collaborative filtering recommender system using learning-to-rank and rank aggregation." Arabian Journal for Science and Engineering 45 (2020): 2835-2845. https://doi.org/10.1007/s13369-019-04180-3
73. Xiaojian Zhao, Guangda Li, Meng Wang, Jin Yuan, Zheng-Jun Zha, Zhoujun Li, and Tat-Seng Chua. "Integrating rich information for video recommendation with multi-task rank aggregation." In Proceedings of the 19th ACM international conference on Multimedia, Scottsdale, USA, November 28-December 1, 2011, pp. 1521-1524. https://doi.org/10.1145/2072298.2072055
74. Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. "LETOR: A benchmark collection for research on learning to rank for information retrieval." Information Retrieval 13 (2010): 346-374. https://doi.org/10.1007/s10791-009-9123-y
75. Olivier Chapelle, and Yi Chang. "Yahoo! learning to rank challenge overview." In Proceedings of the learning to rank challenge, pp. 1-24. PMLR, 2011.
76. Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft. "Unbiased learning to rank with unbiased propensity estimation." In The 41st international ACM SIGIR conference on research & development in information retrieval, USA, July 8 - 12, 2018, pp. 385-394. https://doi.org/10.1145/3209978.3209986

77. Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. "Post-learning optimization of tree ensembles for efficient ranking." In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, Pisa, Italy, July 17-21, 2016, pp. 949-952. https://doi.org/10.1145/2911451.2914763
78. Lixin Zou, Shengqiang Zhang, Hengyi Cai, Dehong Ma, Suqi Cheng, Shuaiqiang Wang, Daiting Shi, Zhicong Cheng, and Dawei Yin. "Pre-trained language model based ranking in Baidu search." In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, Singapore, August 14 - 18, 2021, pp. 4014-4022. https://doi.org/10.1145/3447548.3467147
79. Jun Xu, Zeng Wei, Long Xia, Yanyan Lan, Dawei Yin, Xueqi Cheng, and Ji-Rong Wen. "Reinforcement learning to rank with pairwise policy gradient." In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, China, July 25-30, 2020, pp. 509-518. https://doi.org/10.1145/3397271.3401148
80. Maximilian Mayerl, Michael Vötter, Günther Specht, and Eva Zangerle. "Pairwise learning to rank for hit song prediction." BTW 2023 (2023). 10.18420/BTW2023-26
81. Chiqun Zhang, Michael R. Evans, Max Lepikhin, and Dragomir Yankov. "Fast Attention-based Learning-To-Rank Model for Structured Map Search." In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Canada, July 11-15, 2021 pp. 942-951. https://doi.org/10.1145/3404835.3462904
82. Azam Seilsepour, Reza Ravanmehr, and Ramin Nassiri. "Topic sentiment analysis based on deep neural network using document embedding technique." The Journal of Supercomputing (2023): 1-39. https://doi.org/10.1007/s11227-023-05423-9
83. Samuel EL Oliveira, Victor Diniz, Anisio Lacerda, Luiz Merschmanm, and Gisele L. Pappa. "Is rank aggregation effective in recommender systems? an experimental analysis." ACM Transactions on Intelligent Systems and Technology (TIST) 11, no. 2 (2020): 1-26. https://doi.org/10.1145/3365375
84. Reza Jafari Ziarani, and Reza Ravanmehr. "Serendipity in recommender systems: a systematic literature review." Journal of Computer Science and Technology 36 (2021): 375-396. https://doi.org/10.1007/s11390-020-0135-9
85. Reza Jafari Ziarani, and Reza Ravanmehr. "Deep neural network approach for a serendipity-oriented recommendation system." Expert Systems with Applications 185 (2021): 115660. https://doi.org/10.1016/j.eswa.2021.115660
86. Michał Bałchanowski, and Urszula Boryczka. "A Comparative Study of Rank Aggregation Methods in Recommendation Systems." Entropy 25, no. 1 (2023): 132. https://doi.org/10.3390/e25010132

Summary

Today, due to the increasing volume of data and the human need to quickly access the required information, recommender systems play a crucial role in daily life. Recommender systems try to provide effective suggestions to users that match their personal preferences based on explicit or implicit data extracted from users' actions and behaviors.

One of the important challenges of conventional recommender systems is to focus on the long-term interests of users statically and ignore the patterns of short-term interests of users. Another problem is the unavailability of user information and characteristics due to data privacy and optional user authentication. A session-based recommender system (SBRS) is presented to reduce the effects of the mentioned problems. The recommendation process in session-based recommender systems is based on learning the dependencies within each session or between several sessions, which are recognized based on the co-occurrences of the interactions. Deep learning methods are one of the most extensively employed and essential methods used to correctly detect dependencies between interactions in sessions. This book provides a comprehensive review of diverse methodologies employed in the context of session-based recommender systems that incorporate deep learning techniques.

This book was organized into six chapters as follows:

Chapter 1 reviewed the general concepts of recommender systems to highlight the necessity of session-based recommender systems. Then, the fundamental concepts, data and task modeling challenges, the classification of methods, and an overlook of various approaches of session-based recommender systems were presented. Since the concepts of session-based and sequential recommender systems are close to each other, the rest of this chapter clarifies and distinguishes between their boundaries.

Due to the high importance of deep learning and the diversity of their techniques and applications in different domains, specially in SRS, they were studied briefly in Chap. 2. This chapter presented the history and timeline of deep learning; determined the position of deep learning next to the fields of artificial intelligence, machine learning, and data science; and addressed its advantages and disadvantages.

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2

Then, a taxonomy was presented according to the types of deep learning methods and their use in different research in the scope of session-based recommender system. In this taxonomy, deep learning models were classified into discriminative, generative, and graph-based approaches, and then each of these three categories was further divided into different models and discussed.

Chapter 3 focused on the approaches of session-based recommender systems using deep discriminative models. A review of the literature on SBRS shows that various methods using RNN and its variants (GRU and LSTM) have been proposed to model the dynamic behaviors of session data over time. Due to the sequential nature of session data, many session-based recommender systems use RNNs. Indeed, RNNs have a hidden state with non-linear dynamics that enables them to discover patterns of events and predict the next item. On the other hand, CNN can extract spatial and temporal features and patterns among data, reducing the need for manual feature engineering. For this reason, it is used in session-based recommender system to extract effective patterns from interactions and predict the next items. At the beginning of the chapter, a brief overview of the fundamental concepts of these models, the conventional datasets, and the evaluation methods used were provided. Then, in the following subsections, different methods based on LSTM, GRU, and then CNN were discussed and analyzed separately.

Chapter 4 of the book was devoted to session-based recommender systems using deep generative models. Despite the advantages of using deep discriminative methods in SBRS, these approaches obtain information entropy through the conditional distributions of subsequent clicks relative to previous clicks and typically choose a unimodal or a mixture of unimodal. In sessions, intrinsic structural sequences may lead to mutual influence between different output variables in a time step. Discriminative methods may not have the necessary efficiency for these types of systems and may deviate from the main goals as the sessions expand. Therefore, deep generative models, which are strengthened by examining multimodal output distributions and uncertainty estimation, can be employed for SBRS research. Moreover, deep generative methods can produce more samples for model training and reduce the problems caused by data sparsity. Most of the research in this field is based on autoencoders (AE), generative adversarial networks (GAN), and flow-based models (FBM). At the beginning of the chapter, a brief overview of the fundamental concepts of generative methods, the conventional datasets, and the basic evaluation methods of various research were discussed. In the subsequent sections, different methods based on AEs, GANs, and flow-based methods were investigated.

Due to the high flexibility of deep neural networks and to take more benefit of each method and reduce their limitations, many proposed session-based recommender system utilized an hybrid/advanced deep neural network model. Chapter 5 discussed the session-based recommender systems using hybrid/advanced models. In many cases, due to the sequential nature of user interactions, RNNs are an essential part of the hybrid approach. In addition, the graph neural networks and deep reinforcement learning models in SBRS were also discussed in this chapter.

The ranking methods that give priority to presenting them to the users are essential in all recommender systems. On the other hand, learning to rank (LtR) has emerged based on a combination of machine learning and information retrieval. Chapter 6 was dedicated to learning to rank (LtR) methods in session-based recommender system. LtR methods were discussed in the two fields of ranking creation and ranking aggregation. Ranking creation methods are divided into four categories: pointwise, pairwise, listwise, and hybrid methods. In the remainder of this chapter, ranking aggregation methods were also included. The approaches presented for each of these methods have been investigated both in information retrieval and recommender systems scopes.

In closing, the goal of this book is to help researchers/engineers who are interested in using deep learning techniques in session-based recommender systems. The material of this book will help readers to simultaneously enhance their knowledge in the two contexts of session-based recommender systems and deep learning. To this end, the book presents a comprehensive overview of the methods presented in session-based recommender systems from various aspects, which provides a fundamental technical background for developing these types of systems.

Index

R. Ravanmehr, R. Mohamadrezaei, *Session-Based Recommender Systems Using Deep Learning*, https://doi.org/10.1007/978-3-031-42559-2