People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

IBN KHALDOUN UNIVERSITY OF TIARET

Dissertation

Presented to:

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTEMENT OF COMPUTER SCIENCE

In order to obtain the degree of:

MASTER

Specialty: Software Engineering

Presented by:

BELHOCINE KHEIRA AMEL
GUELFOUT AMEL

On the theme:

Session-based Recommender Systems Using Graph Convolutional Networks

---

Defended publicly on: 09 / 07 /2023 in Tiaret in front the jury composed of:

Mr. CHIKHAOUI Ahmed	MCB	Tiaret University	Chairman
Mr. BOUDAA Boudjemaa	MCA	Tiaret University	Supervisor
Mme. LAAREDJ Zohra	MAA	Tiaret University	Examiner

2022-2023

Abstract

The rapid growth of online platforms has led to an overwhelming amount of available information and choices, making personalized recommendations crucial for enhancing user experience and satisfaction. In this thesis, we focus on session-based recommender systems, which aim to provide accurate recommendations by considering users' sequential behaviour and short-term interests.

To address the challenges posed by session-based recommendations, we leverage the power of Graph Convolutional Networks (GCNs). GCNs have shown remarkable effectiveness in modelling complex relationships and capturing the underlying structure of data. By exploiting the graph-like nature of user sessions, we harness the potential of GCNs to capture the intricate dependencies between items and uncover latent patterns within sessions.

Keywords: Recommender systems, Session-based recommendations, Graph Convolutional Networks, Sequential behaviour, Personalization.

الملخص

نمت الأنظمة العاملة عبر الإنترنت بسرعة، مما أدى إلى توفر كمية هائلة من المعلومات والخيارات المتاحة، مما يجعل التوصيات الشخصية أمرًا حاسمًا لتعزيز تجربة المستخدم ورضاه. في هذه الأطروحة، نركز على أنظمة التوصية القائمة على الجلسات، التي تهدف إلى تقديم توصيات دقيقة من خلال النظر في سلوك المستخدمين التسلسلي واهتماماتهم القصيرة المدى.

للتغلب على التحديات التي تطرحها التوصيات القائمة على الجلسات، نستغل قوة شبكات التكرار الرسومية (GCNs). أظهرت شبكات التكرار الرسومية فعالية ملحوظة في نمذجة العلاقات المعقدة واستخلاص الهيكل الأساسي للبيانات. من خلال استغلال طبيعة الجلسات الشبيهة بالرسوم البيانية للمستخدمين، نستخدم إمكانات شبكات التكرار الرسومية لالتقاط التبعيات المعقدة بين العناصر واكتشاف الأنماط الكامنة داخل الجلسات.

الكلمات المفتاحية: أنظمة التوصية، التوصيات القائمة على الجلسات، شبكات التكرار الرسومية، السلوك التسلسلي، التخصيص الشخصي.

Acknowledgments

Foremost, praises and thanks to Allah, the Almighty, for His showers of blessings throughout my research work, which allowed me to complete it successfully.

I would like to extend my sincere gratitude to my thesis advisor, Mr. Boudaa Boudjemaa. His unwavering support and expertise have been invaluable throughout my research. I am truly grateful for his patience, mentorship, and the freedom he granted me to explore and develop my ideas.

I am deeply indebted to my mom, my dad and my aunt, whose constant support has been a pillar of strength throughout my years of study and in every aspect of my life. Their unwavering belief in my abilities, their encouragement, their love, guidance, and their sacrifices has played a vital role in my achievements.

I would also like to acknowledge my binome, Amel. This work wouldn't have been accomplished without your hard work and dedication.

To all those who have supported me, teachers, innovation team, friends, or followers, I extend my deepest thanks. Your contributions, whether big or small, have left an indelible mark on my life. Your presence has been instrumental in shaping my character.

This accomplishment would not have been possible without you. Thank you all.

Belhocine Kheira Amel

First and foremost, I am deeply thankful to Allah for His guidance, blessings, and reconciliation in my life.

To my family, especially my parents, I would like to offer a special dedication for everything they have given me.

I would like to express my utmost respect and appreciation to my supervisor, Mr. Bouadaa Boudjemaa. Your guidance, expertise, and mentorship have been instrumental in shaping this thesis.

I would also like to acknowledge my binome, Nesrine. This work wouldn't have been accomplished without your hard work and dedication.

To my university professors who have supported me throughout my journey, I am incredibly grateful. You have provided me with invaluable knowledge and experiences that words cannot express.

Lastly, I want to express my gratitude to all those names may not be mentioned here but have played a part in my academic and personal growth. Your friendship, encouragement, and positive influence have been invaluable.

Thank you, each and every one of you, for your profound impact on my life and for being an integral part of this remarkable journey.

Guelfout Amel

Table of Content

List of Figure

List of tables

Acronyms

ANN Artificial Neural Network

AI Artificial Intelligence

CARS Context-aware Filtering

CB Content-based Filtering

CF Collaborative Filtering

CNN Convolutional Neural Network

DL Deep Learning

FNN Feed-forward Neural Network

GCN Graph Convolutional Network

GNN Graph Neural Network

GNN Graph Neural Network

ML Machine Learning

RNN Recurrent Neural Network

RS Recommender System

SBRS Session-based Recommender System

General Introduction

A. Background

In recent years, the rapid growth of online platforms and the abundance of available information have presented both opportunities and challenges for recommender systems. Recommender systems play a crucial role in assisting users in navigating through vast amounts of content to find relevant and personalized recommendations. Traditional recommender systems often focus on item-to-item or user-to-user interactions, neglecting the temporal nature of user behaviour. However, users' preferences and interests are not static; they evolve over time, and capturing this temporal aspect is essential for delivering accurate recommendations.

B. Problem Statement

The aim of this thesis is to address the limitations of existing recommender systems by proposing a session-based approach that incorporates the concept of Graph Convolutional Networks (GCN). Session-based recommender systems aim to capture user behaviours within a specific session, considering the sequence of items accessed by users and the temporal dependencies between them. By utilizing GCN, which has demonstrated remarkable performance in various graph-based tasks, we can leverage the rich structural information available in user-item interaction graphs to enhance the quality of recommendations.

C. Delimitation

The challenge we interfere is to capture the dynamic nature of user preferences and interests, as well as modelling the complex relationships between items within a session. By incorporating GCN, we aim to overcome this delamination problem and develop a session-based recommender system that can effectively exploit the temporal and structural information present in user-item interaction graphs.

D. Approach

To achieve our objective, we will adopt a two-step approach. First, we will collect and pre-process a large-scale dataset of user-item interactions, capturing the temporal dynamics of user behaviour within sessions. Next, we will design and implement a session-based recommender system that utilizes GCN to model the sequential dependencies and item relationships within sessions. We will evaluate the performance of our proposed system using various metrics such as accuracy, recall, and mean average precision, comparing it against existing state-of-the-art recommender systems.

E. Outline

The remainder of this thesis is organized into three chapters besides a general introduction and a general conclusion:

- ***General Introduction***

  An initiation to recommender system, the background, problem statement, and the delamination of those papers.

- ***Chapter one: Session-based Recommender Systems***

  An overview of recommender systems and their types, going into session-based recommender systems that consider as the core research of our thesis.

- ***Chapter two: Graph Convolutional Networks***

  Essential background of artificial intelligence and its sub-fields, digging into neural networks and their architecture mentioning an overview and architecture of its standard ANNs types, in the reasons of giving a solid understanding before going into graph convolutional networks

- ***Chapter three: Developing a Session-based Recommender Systems using Graph Convolutional Networks***

  Present the model proposed for developing a session-based recommender systems using graph convolutional networks

- ***General Conclusion***

  Based on the results and the discussion in the previous chapter, this last part relates to the research questions and draws a summary of this work. Finally, suggestions for potential future work are discussed.

Chapter 1. Session-Based Recommender Systems

1.1. Introduction

Session-based recommender systems (SBRSs) are a type of recommender systems that make recommendations based on users' short-term interests and preferences. They are becoming popular in domains such as e-commerce, music streaming, and news recommendation. SBRSs are challenging to build due to issues such as data sparsity, short-term user behaviour, and the lack of explicit user feedback. In this chapter, we will provide an overview of RSs, including their definition, types and challenges. Finally, we will provide a detailed discussion of SBRSs, which is a type of recommender systems; we will include its definition, sessions and session's properties and its limitations and challenges.

1.2. Recommender Systems

Recommender Systems (RSs) have been largely studied for the past decade and have shown to be suitable for many scenarios. On the arrival of the internet and the era of e-commerce, companies are opting for having a RS as an attempt to boost sales. RSs provide predictions of items that the user may find interesting to purchase [1], in which most algorithms for this purpose focus on providing recommendations that fit the preferences of the user [2].

RSs have shown to be useful for users and business. Users suffer from what is called the paradox of choice. Having many options to choose from lead to more difficulty in effectively making a choice [3]. They provide suggestions for items[1] that are of potential interest for a user. These systems are applied for answering questions such as which book to buy? , which website to visit next? , and which financial service to choose? » [4] They are widely used in various applications, such as e-commerce, social media, and entertainment platforms, to enhance user experience, increase engagement, and drive sales. Recommender systems have the effect of guiding users in a personalized way to interesting or useful objects in a large space of possible options [4]. They have evolved into a fundamental tool for making more informative, efficient and effective choices and decisions in almost every daily aspect of life, working, business operations, study, entertaining and socialization. Their roles have become ever important in the increasingly overloaded age of digital economy where users have to make choices from usually massive and rapidly increasing contents, products and services (which are uniformly called items) [5]. Therefore an RS can be seen as a system [6, 7], which consists of multiple basic entities including *users*, *items* and their *behaviours*, e.g., user-item interactions [5].

There are plenty of examples of companies that use RSs. For instance, Amazon and many e-commerce have adopted the use of recommendation engines. Other services such as Netflix, YouTube and Last FM also use recommender systems [2].

[1] Is the general term used to denote what the system recommends to users [4]

1.3. Overview of Recommender Systems Types

Recommender systems can be broadly classified into several categories based on the approach used to generate recommendations, its main approach shown in Figure 1 [8].

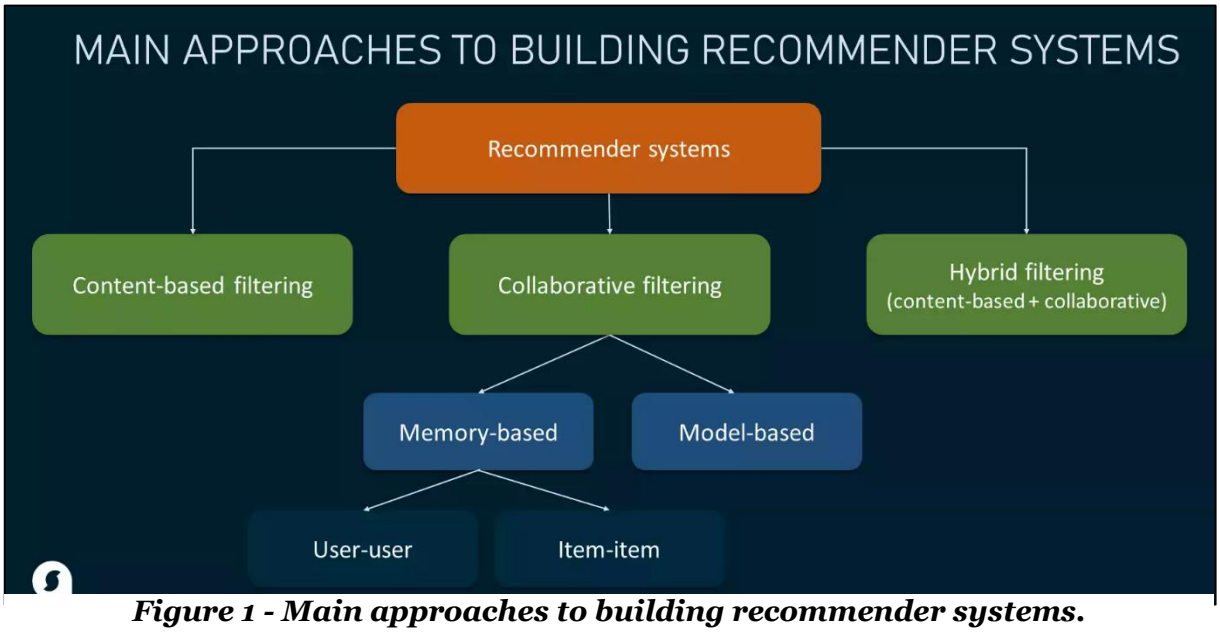


Figure 1 - Main approaches to building recommender systems.

1.3.1. Content-based filtering

A content-based recommender system suggests those items that are similar in features to items user has already liked in past [9]. A typical CB recommender first creates user profile using user feedback and ratings about items. The user profile is then compared with item features and the matched items are recommended [10]. Such systems are used in recommending web pages, TV programs and news articles etc. [11].

To understand this, let's use a simple example took from [12] shows how a content-based recommender system might work to suggest movies.

Let's suppose there are four movies and a user has seen and liked the first two (see Figure 2).

The model automatically suggests the third movie rather than the fourth, since it is more similar to the first two. This similarity can be calculated based on a number of features like the actors and actresses in the movie, the director, the genre, the duration of the film, etc.

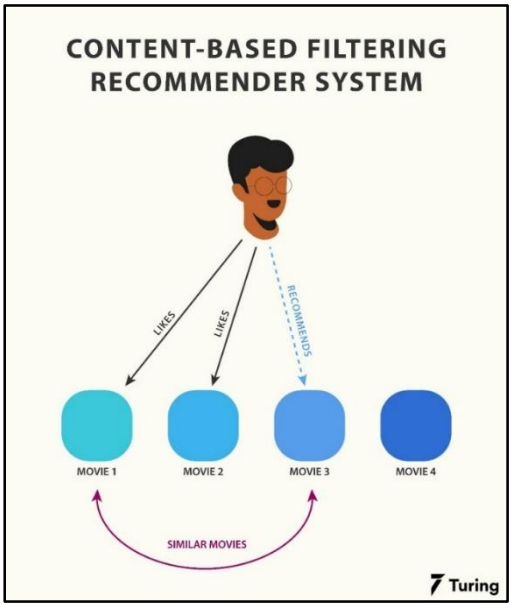


Figure 2 - Content-based filtering example

1.3.2. Collaborative-filtering

As shown in Figure 3 [13] ,collaborative filtering recommends items by matching users with other users having similar interests [14]. It collects user feedback in the form of ratings provided by user for specific item and finds match in rating behaviours among users in order to find group of users having similar preferences. Here, a user profile represents user preferences that the user has either explicitly or implicitly provided [10].

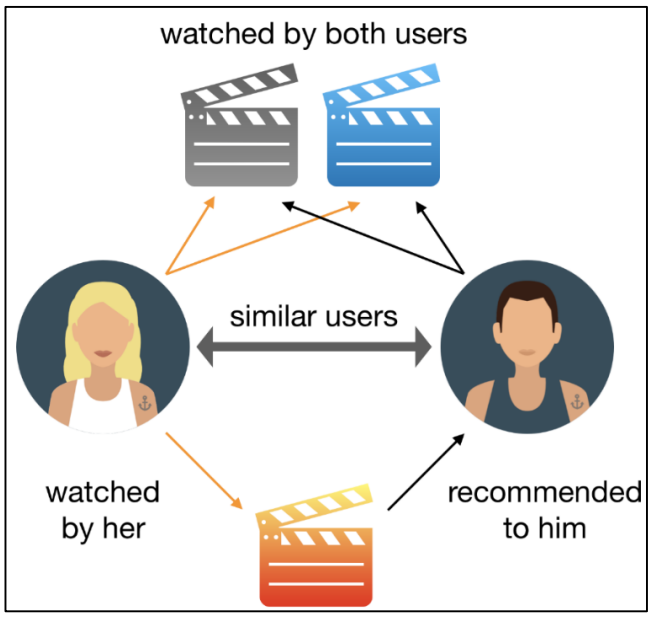


Figure 3 - Collaborative-filtering example

1.3.3. Hybrid-filtering

In hybrid approach, two or more filtering methods are combined to gain better performance over CB[2] and CF[3] approaches when they are applied separately. Several researchers combined CB and CF techniques for gaining better results and mitigating various shortcomings faced by these approaches [10]. Burke [15] has categorized hybridization methods into seven different types including: (1) weighted, (2) switching, (3) mixed approach, (4) feature combination, (5) cascade, (6) feature, and (7) meta-level hybridization approach.

1.3.4. Knowledge-based recommender systems

A recommender system is knowledge-based when it makes recommendations based not on a user's rating history, but on specific queries made by the user. It might prompt the user to give a series of rules or guidelines on what the results should look like, or an example of an item. The system then searches through its database of items and returns similar results [16].

1.3.5. Demographic-based filtering

Demographic Recommender system generates recommendations based on the user demographic attributes. It categorizes the users based on their attributes and recommends the movies by utilizing their demographic data [4]. In contrast to collaborative filtering and content-based recommender system, it is easy to implement and does not require user ratings [17].

1.3.6. Context-aware filtering (CARS)

Context is a complex notion that has been studied across different research disciplines [18]. The definition introduced in [19] has been widely adopted for CARS and states the following: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application*". Consequently, recommender systems produce suggestions by leveraging previously mentioned context. This methodology incorporates up-to-date data regarding the user's present circumstances, thereby enhancing the relevancy of the recommendations generated.

1.4. Evaluation Metrics for Recommender Systems

The evaluation of recommender systems is a fundamental aspect that aims to measure how effectively they provide personalized recommendations to users based on their preferences and needs. This section discusses the evaluation metrics and used to assess the performance of recommender systems.

1.4.1. Predictive Accuracy Metrics

Predictive accuracy or *rating prediction* metrics embark on the question of how close the ratings estimated by a recommender are to the true user ratings. This type of measures

[2] Content-based Filtering
[3] Collaborative Filtering

is very popular for the evaluation of non-binary ratings. It is most appropriate for usage scenarios in which an accurate prediction of the ratings for all items is of high importance [20].

1.4.2. Mean reciprocal rank (MRR)

According to [21] MRR is the average of reciprocal rank (RR) over users. The reciprocal rank is the "multiplicative inverse" of the rank of the first correct item. MRR is an appropriate choice in two cases:

- There is only one relevant item.
- In the use case, only the first recommended item is the essential one.

It means that MRR doesn't apply if there are multiple correct responses in the resulting list. If your system returns 10 items and it turns out there is a relevant item in the third-highest spot, that's what MRR cares about. It will not check if the other relevant items occur between rank 4 and rank 10.

1.4.3. Novelty Metrics

It is a measure of the ability of RS to introduce long-tail items to users. E-commerce platforms can benefit from high-ranking individualized, niche items. For example, Amazon makes a great success by selling books that are not available in traditional book stores, rather than bestsellers.

Novelty can be defined as a fraction of unknown items among all items the user liked. An ideal way of measuring it would be a customer survey but in most cases, we are unable to determine whether the user knew the item before. Having implicit data about user behaviours allows us to measure dissimilarity between recommendations that sometimes substitutes novelty scores. We have to also remember that too many novel items can result in a lack of trust from users. It is essential to find the balance between novelty and trustworthiness [21].

1.4.4. Diversity Metrics

Diversity is a concept concerned with the diversity of items in the recommendation list [2]. It is a measure of how your recommendations are different from each other. Consider the customer finished watching the first movie of a trilogy on Netflix. Low diversity recommender would recommend only the next parts of the trilogy or the films directed by the same director. On the other hand, high diversity can be achieved by recommending items completely random [22].

1.4.5. Serendipity Metrics

Serendipity represents surprising recommendations. *Iaquinta et al*[4]. [23], mention that serendipity represent items that the users would difficultly find. It can be concluded that even though serendipity has *a hard-to-understand definition*, most authors agree that it

[4] is an abbreviation used in academic writing to refer to a group of authors.

represents a delightful surprise and provide useful and surprising items to the user [2]. Common serendipity metrics include mean average precision and unexpectedness.

1.4.6. Coverage Metrics

Coverage is the ability of the recommender system to recommend all items from a train set to users. Let's consider the random recommender that selects items as in the lottery drawing. Such recommender has nearly 100% coverage because it has the ability to recommend every available item. On the other hand, the popularity-based recommender is going to recommend just top k items. In such a case, coverage is close to 0%.

Coverage does not evaluate if the user enjoys the recommendation or not, instead, it measures the RS in terms of its ability to bring unexpectedness to the user. Low coverage can lead to users' dissatisfaction [21].

1.5. Experimental Settings

In this section, we discuss three levels of experiments for comparing multiple recommenders. We start with offline experiments, which are relatively easier to conduct as they do not require interaction with real users. Next, we describe user studies, where a small group of subjects is asked to use the system in a controlled environment, and their experiences are reported. These experiments provide both quantitative and qualitative information about the systems, but it is essential to consider various biases in the experimental design.

Lastly, the most reliable type of experiment involves the system being used by a pool of real users, who are typically unaware of the experiment. Although this type of experiment allows us to collect only specific types of data, it closely resembles real-world conditions and provides valuable insights [24], those are used in recommender systems to evaluate and compare the performance of different recommendation algorithms or approaches.

1.5.1. Offline Experiments

An offline experiment is performed by using a pre-collected data set of users choosing or rating items. Using this data set we can try to simulate the behaviours of users that interact with a recommendation system. In doing so, we assume that the user behaviours when the data was collected will be similar enough to the user behaviours when the recommender system is deployed, so that we can make reliable decisions based on the simulation. Offline experiments are attractive because they require no interaction with real users, and thus allow us to compare a wide range of candidate algorithms at a low cost [24].

1.5.2. User Studies

A user study is conducted by recruiting a set of test subjects, and asking them to perform several tasks requiring an interaction with the recommendation system. While the subjects perform the tasks, we observe and record their behaviours, collecting any number of quantitative measurements, such as what portion of the task was completed, the accuracy of the task results, or the time taken to perform the task [24].

1.5.3. Online evaluation

Online evaluation is one of the best ways of seeing user interactions with the recommendation engine. The real-life performance of the recommendation system depends on a variety of factors. During the online evaluation, real users interact with the systems. So, it is possible to understand the correct user intent and the success of the recommendation model directly [22].

1.6. Limitations and Challenges of Recommender Systems

1.6.1. Cold Start

The “cold start" problem happens in recommendation systems due to the lack of information, on users or items. The Cold-Start problem is a well-known issue in recommendation systems: there is relatively little information about each user, which results in an inability to draw inferences to recommend items to users [11] ,i.e., If you are building a brand-new recommendation system, you would have no user data to start with. You can use content-based filtering first and then move on to the collaborative filtering approach [25].

1.6.2. Scalability

The rate of growth of nearest-neighbour algorithms shows a linear relation with number of items and number of users. It becomes difficult for a typical recommender to process such large-scale data [10]. Different techniques have been proposed including clustering, reducing dimensionality, and Bayesian Network [26].

1.6.3. Data Sparsity

In practice, many commercial recommender systems are based on large datasets. As a result, the user-item matrix used for collaborative filtering could be extremely large and sparse, which brings about the challenges in the performances of the recommendation. One typical problem caused by the data sparsity is the cold start problem. As collaborative filtering methods recommend items based on users’ past preferences, new users will need to rate a sufficient number of items to enable the system to capture their preferences accurately and thus provide reliable recommendations. Similarly, new items also have the same problem. When new items are added to the system, they need to be rated by a substantial number of users before they could be recommended to users who have similar tastes. The new item problem does not limit the content-based recommendation because the recommendation of an item is based on its discrete set of descriptive qualities rather than its ratings [11].

1.6.4. Synonymy

Synonymy arises when an item is represented with two or more different names or entries having similar meanings [27]. In such cases, the recommender cannot identify whether the terms represent different items or the same item.

1.6.5. Shilling attacks

What happens if a malicious user or competitor enters into a system and starts giving false ratings on some items either to increase the item popularity or to decrease its popularity [28]. Such attacks can break the trust on the recommender system as well as decrease the performance and quality of recommenders [10].

1.6.6. Privacy Concern

The more the algorithm knows about the customer, the more accurate its recommendations will be. However, many customers are hesitant to hand over personal information, especially given several high-profile cases of customer data leaks in recent years. However, without this customer data, the recommendation engine cannot function effectively. Therefore, building trust between the business and customers is key [29].

1.7. Session-based Recommender Systems

1.7.1. Overview

For present e-commerce platforms, it is important to accurately predict users' preference for a timely next-item recommendation. To achieve this goal, session-based recommender systems are developed, which are based on a sequence of the most recent user-item interactions to avoid the influence raised from outdate historical records [30] . An SBRS aims to predict either the *unknown* part (e.g., an item or a batch of items) of a session given the *known* part, or the *future session* (e.g., the next-basket) given the historical sessions via learning the intra- or inter-session dependencies [5].

SBRSs learn users' preferences from the *sessions* associated and generated during the consumption process. Each *session* is composed of multiple *user-item* interactions that happen together in a *continuous* period of time, e.g., a basket of products purchased in one transaction visit, which usually lasts for several minutes to several hours [5]

1.7.2. Session and Session Properties

A session refers to a sequence of user engagements with an application or website within a brief timeframe. These engagements encompass various actions like clicks, views, purchases, searches, and more. Each session is linked to specific session properties, including the items viewed, the duration spent on each item, the sequence of item views, and the time gaps between consecutive interactions.

A session can usually reflect a user's current preference, a local shift of the user's intention within the session may still exist [30]. It is a list of interactions with a clear boundary [5].

We will discuss five important properties of sessions that have a great impact on SBRSs:

Property 1: session length. The length of a session is defined as the total number of interactions contained in it [5]. This is a basic property of sessions, which is taken as one of

the statistical indicators of experiment data in most literature [31, 32]. Sessions of different lengths may bring different challenges for SBRSs and thus lead to different recommendation performance. The session characteristics related to session length together with the corresponding challenges for building SBRSs are discussed in detail in Section 1.7.4.

Property 2: internal order. The internal order of a session refers to the order over interactions within it. Usually, there are different kinds of order flexibility inside different sessions. The existence of internal order leads to the sequential dependencies within sessions which can be used for recommendations [5]. The session characteristics related to internal order and its challenges for building SBRSs are discussed in detail in Section 1.7.4.

Property 3: action type. In the real world, some sessions contain only one type of actions. The dependencies over different types of actions are often different. For instance, the items that are clicked together in a session may be similar or competitive while the items purchased together in one session may be complementary. Therefore, the number of action types in a session determines whether the intra-session dependencies are *homogeneous* (based on a single type of actions) or *heterogeneous* (based on multi-type actions), which is important for accurate recommendations [5]. The session characteristics related to action type as well as the corresponding challenges for building SBRSs are discussed in detail in Section 1.7.4.

Property 4: user information. User information of a session mainly refers to the IDs of the users in the session, and sometimes user attributes are also included. In this paper, the property of user information refers to the availability of user information in a session. In the real word, the user information of sessions is given in some cases, while it is not available in other cases [33, 34, 35] User information plays an important role to connect sessions from the same user happening at different time and thus its availability determines the possibility to model the long-term personalized preference across multiple sessions for a specific user. In practice, SBRSs were initially proposed to handle those anonymous sessions where user information is not available [36]. The session characteristics together with the corresponding challenges for building SBRSs are discussed in detail in Section 1.7.4.

Property 5: session-data structure. Session-data structure refers to the session-related hierarchical structure consisting of multiple levels [37, 38], which intrinsically exists in some session data. The interaction level is necessary for a session, while the other levels depend on the specific session data. This is because either the attribute information or the historical session information may not be available in all session data. Usually, the number of levels included in a session data set determines the information volume that can be used for recommendations [5]. The session characteristics related to session-data structure as well as the corresponding challenges for building SBRSs are discussed in detail in Section 1.7.4.

1.7.3. Sub-area of SBRS

The variety of existing work on SBRSs can be generally categorized into three sub-areas fitting a unified categorization framework to reduce the aforementioned inconsistencies

and confusion. According to the difference on the recommendation tasks, the sub-areas include *next interaction recommendation*, *next partial-session recommendation*, and *next session recommendation* [5]. Table 1 illustrates the differences between sub-area in SBRSs [5]

- ***Next interaction recommendation.*** Next interaction recommendation aims to recommend the next possible interaction in the current session by mainly modelling intra-session dependencies [5].
- ***Next partial-session recommendation.*** Next-partial session recommendation aims to recommend all the remaining interactions to complete the current session, e.g., to predict all the subsequent items to complete a basket given the purchased items in it, by mainly modelling intersession dependencies [5].
- ***Next session recommendation.*** Next session recommendation aims to recommend the next session, e.g., next basket, by mainly modelling inter-session dependencies [5].

Table1- A comparison of different sub-areas in SBRSs

Sub-area	Input	Output	Typical research topic
Next interaction recommendation	Mainly known part of the current session	Next interaction (item)	Next item recommendation, next song/movie recommendation, next POI recommendation, next web page recommendation, next news recommendation, etc.
Next partial-session recommendation	Mainly known part of the current session	Subsequent part of the session	Next items recommendation, session/basket completion
Next session recommendation	Historical sessions	Next session	Next basket recommendation, next bundle recommendation, etc.

1.7.4. Characteristics and Challenges

According to [2], gaining a comprehensive understanding of the characteristics of session data and the challenges associated with modelling it is crucial in order to develop a well-suited Session-Based Recommender System (SBRS). In this section, we illustrate and summarize these characteristics and challenges as follows:

- **Related to session length**

According to session length, sessions can be roughly categorized into three types: long sessions, medium sessions and short sessions.

Long sessions: A session that is considered long typically consists of a higher number of interactions, exceeding 10 or more. In essence, longer sessions tend to offer a greater amount of contextual information, which can contribute to the generation of more precise recommendations. However, due to the uncertainty of user behaviours', a long session is more likely to contain random interactions [39] which are not related or unrelated to other interactions within it. This brings noisy information and thus reduces the performance of recommendations [40, 41].Therefore; the first challenge for SBRSs built on long sessions is how to effectively reduce the noisy information from the irrelevant interactions [5]. In

addition, there are usually more complex dependencies embedded in a long session, e.g., long-range dependencies [42] between two interactions that are far from each other in a session or high-order dependencies [41]across multiple interactions in a session. Consequently, another challenge for SBRSs built on long sessions is how to effectively learn complex dependencies for better recommendation performance [5].

Medium session: Sessions of medium duration tend to encompass a moderate number of interchanges, ranging from approximately 4 to 9 interactions. When compare medium sessions with long and short sessions, a medium session is less likely to contain too many irrelevant interactions while it usually contains the necessary contextual information for Session-Based Recommendation (SBR) [5]. Although less complex in nature, the development of SBRSs for medium length sessions remains fundamentally challenged.

Short sessions*:* It consists of limited interactions that are usually less than 4, consequently limiting the information available to substantiate recommendations**.**

- **Related to internal order**

Sessions can be divided into unordered sessions, ordered sessions and flexible-ordered sessions:

Unordered sessions: An unordered session contains interactions without any chronological order between them, namely, whether an interaction happens earlier or later in the session makes no difference [31]. For example, the shopping sessions are sometimes unordered since users may pick up a basket of items (e.g., {bread, milk, eggs}) without following an explicit order [32]. In unordered sessions, the dependencies among the interactions are based on their co-occurrence rather than the sequences of them, and thus the generally utilized sequence models are not applicable [5]. Furthermore, most of co-occurrence-based dependencies among interactions are collective dependencies [43, 44].

Ordered sessions*:* An ordered session contains multiple interactions with strict order, and usually strong sequential dependencies exist among them. Although it is relatively easy to learn the strong sequential dependencies within ordered sessions, it is challenging *to effectively learn the cascaded long-term sequential dependencies which decay gradually with time in long ordered sessions* [5].

Flexibly-ordered sessions: A flexibly-ordered session is neither totally unordered nor totally ordered, i.e., some parts of the session are ordered while others are not [43]. Therefore, the complex dependencies inside flexibly-ordered sessions must be carefully considered and precisely learned for accurate recommendation. Consequently, the challenge for SBRSs built on flexibly-ordered sessions comes from how to effectively learn the complex and mixed dependencies [5].

- **Related to action type**

Sessions can be divided into single-type-action sessions and multi-type-action sessions:

***Single-type-action sessions**:* A single-type-action session includes one type of actions only, e.g., clicks of items, and thus only one type of dependencies comes from the same type of actions, which is relatively easy to learn [5].

Multi-type-action sessions: A multi-type-action session includes more than one types of actions [45], leading to multiple types of interactions. Thus, there are complex dependencies inside a multi-type-action session [46]. Specifically, dependencies not only exist over the interactions from the same type (e.g., clicks of items), but also exist over interactions from different types (e.g., clicks and purchases) [5].

Related to user information

According to [5] sessions can be divided into non-anonymous sessions and anonymous sessions.

Non-anonymous sessions: A non-anonymous sessions contains non-anonymous interactions with the associated user information, which enables the connections of different sessions generated by the same user at different time. This makes it possible to learn the user's long-term preference as well as its evolution across sessions. However, due to the relative long time-span and preference dynamics, it is quite challenging *to precisely learn the personalized long-term preference over multiple non-anonymous sessions* [5].

Anonymous sessions: In anonymous sessions, due to the lack of user information to connect multiple sessions generated by the same user, it is nearly impossible to collect the prior historical sessions for the current session. As a result, only the contextual information from the current session can be used for recommendations. Therefore, it is challenging to *precisely capture the user's personalized preference with limited contextual information to provide accurate recommendation* [5].

- **Related to session-data structure**

According to the number of levels of structures, session data can be roughly divided into single-level session data and multi-level session data:

Single-level session data: A single-level session data set is usually a set of anonymous sessions where each consists of several interactions without attribute information or historical session information. In such a case, only single-level dependencies, i.e., the inter-interaction dependencies within sessions can be utilized for recommendations [5]. Hence, due to the lack of auxiliary information from other levels, SBRSs built on single-level session data may easily suffer from the cold-start or data sparsity issue [46]. This leads to the challenge of *how to overcome the cold-start and sparsity issues for accurate recommendations when only the inter-interaction dependencies are available* [5].

Multi-level session data: multi-level session data involves a hierarchical structure of at least two levels, i.e., the interaction level plus attribute level and/or session level. In this case, both the dependencies within each level and across different levels would affect the subsequent recommendations. For example, the categories (the attribute level) of several items may have impact on whether these items would be bought together (the

interaction level) in one session. Consequently, *how to comprehensively learn the intra- and inter-level dependencies for effective and accurate recommendations* becomes a key challenge for SBRSs built on multi-level session data [5].

1.8. Conclusion

In this chapter, we provided an overview of recommender systems and their types. We also discussed various evaluation metrics for recommender systems. Furthermore, we highlighted some of the main limitations and challenges facing recommender systems. In addition, we discussed the concept of session-based recommender systems (SBRSs), by providing an overview of SBRSs. Moreover, we highlighted some of the limitations and challenges associated with SBRS. Overall, this chapter aimed to provide a comprehensive introduction to recommender systems and SBRSs, highlighting their potential and challenges. In the following chapters, we will focus on Graph convolutional networks (GCNs).

Chapter 2.
Graph Convolutional Networks

2.1. Introduction

In this chapter we provide an introduction to Graph Convolutional Networks (GCNs). Which is a powerful tool for analysing graph-structured data in machine learning and data analysis? It begins by establishing the foundational concepts of graphs, learning processes, and neural networks. Then, it introduces Graph Neural Networks (GNNs) as a precursor to understanding GCNs. The core focus is on GCNs, explaining their definition, architecture, types, and applications. We conclude it by addressing the challenges and future research directions in the field of GCNs.

2.2. Deep learning

2.2.1. Overview

In this section, we will explore the fascinating field of deep learning, which lies at the intersection of artificial intelligence (AI) and machine learning (ML). Before delving into the intricacies of deep learning, let's briefly define these foundational concepts:

Artificial Intelligence (AI) as Jair Ribeiro said « AI is the field of computer science that enables machines to perform tasks requiring human-like intelligence. It involves creating intelligent agents that can sense, comprehend, learn, and act in a way that extends human capabilities »

Machine Learning (ML): Machine learning is a subset of AI that focuses on developing algorithms and models that enable computers to learn from data and make predictions or decisions without being explicitly programmed. ML algorithms learn from examples and iteratively improve their performance as they encounter more data. It has become a crucial tool for solving complex problems and driving advancements in various domains.

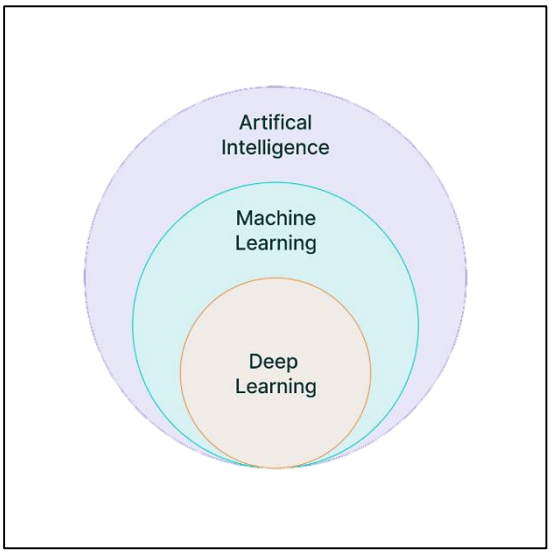


Figure 4 - the relation between AI, ML and Deep Learning

2.2.2. *Learning processes*

There are three major learning paradigms; supervised learning, unsupervised learning and reinforcement learning. Usually, they can be employed by any given type of artificial neural network architecture [47]. The table 2.1 shows a comparison between the three learning paradigms [48].

Supervised learning: Supervised learning is a machine learning technique that sets the parameters of an artificial neural network based on training data. The objective of the learning process is for the artificial neural network to determine the appropriate parameter values for any valid input, having observed the corresponding output values. The training data consists of pairs of input and desired output values, typically represented as data vectors. Supervised learning can also be referred to as classification, where a variety of classifiers exist, each with its own strengths and weaknesses. Selecting a suitable classifier, such as Multilayer Perceptron, Support Vector Machines, k-nearest neighbor algorithm, Gaussian mixture model, Gaussian, naive Bayes, decision tree, radial basis function classifiers, etc., for a given problem is often more of an art than a science [47].

Unsupervised learning: Unsupervised learning is a machine learning technique that sets parameters of an artificial neural network based on given data and a cost function, which is to be minimized. The cost function can be any function and is determined by the task formulation. Unsupervised learning is frequently employed in estimation problems such as statistical modelling, compression, filtering, blind source separation, and clustering. In unsupervised learning, the objective is to determine the organizational structure of the data. It differentiates itself from supervised learning and reinforcement learning in that the artificial neural network is provided with only unlabelled examples. One common form of unsupervised learning is clustering, where the goal is to categorize data into different clusters based on their similarity. Among the aforementioned artificial neural network models, self-organizing maps are the most commonly used unsupervised learning algorithms [47].

Reinforcement learning: Reinforcement learning is a subfield of machine learning that involves an agent learning to interact with an environment to maximize cumulative rewards. The agent learns by taking actions in the environment, receiving feedback in the form of rewards or penalties, and updating its strategy based on this feedback. The ultimate objective is to discover an optimal policy that maps states to actions, leading to the highest expected long-term reward [49, 50].

Table2 - comparisons between the learning paradigms

Criteria	Supervised ML	Unsupervised ML	Reinforcement ML
Definition	Learns by using labelled data	Trained using unlabelled data without any guidance.	Works on interacting with the environment
Type of data	Labelled data	Unlabelled data	No – predefined data
Type of problems	Regression and classification	Association and Clustering	Exploitation or Exploration
Supervision	Extra supervision	No supervision	No supervision
Algorithms	Linear Regression, Logistic Regression, SVM, KNN etc.	K – Means, C – Means, Apriori	Q – Learning, SARSA
Aim	Calculate outcomes	Discover underlying patterns	Learn a series of action
Application	Risk Evaluation, Forecast Sales	Recommendation System, Anomaly Detection	Self Driving Cars, Gaming, Healthcare

2.3. Artificial Neural Networks

2.3.1. Overview

Artificial Neural Networks (ANN) is algorithms based on brain function and are used to model complicated patterns and forecast issues. The Artificial Neural Network (ANN) is a deep learning method that arose from the concept of the human brain Biological Neural Networks. The development of ANN was the result of an attempt to replicate the workings of the human brain. The workings of ANN are extremely similar to those of biological neural networks as shown in Figure 2.2 [51], although they are not identical. ANN algorithm accepts only numeric and structured data [52].

Machine learning is the research field of allowing computers to learn to act appropriately from sample data without being explicitly programmed. Deep learning is a class of machine learning algorithms that is built upon artificial neural networks.

In fact, most of the vital building components of deep learning have existed for decades, while deep learning only gains its popularity in recent years [53], From the realm of Machine Learning and Deep Learning emerges the powerful concept of Artificial Neural Networks (ANNs), which are a machine learning method evolved from the idea of simulating the human brain [54].

In case of biological neuron information comes into the neuron via dendrite, soma processes the information and passes it on via axon. In case of artificial neuron the information comes into the body of an artificial neuron via inputs that are weighted (each

input can be individually multiplied with a weight). The body of an artificial neuron then sums the weighted inputs, bias and "processes" the sum with a transfer function. At the end an artificial neuron passes the processed information via output(s). Benefit of artificial neuron model simplicity can be seen in its mathematical description below [47]:

$$y(k) = F(\sum_{i=0}^{m} w_i(k).x_i(k) + b)$$

Where:

- $x_i(k)$ is the input value in discrete time, where goes from 0 to m,
- $w_i(k)$ is the weight value in discrete time, where goes from 0 to m,
- B is the bias,
- F is the transfer or activation function,
- $y_i(k)$ is the output value in discrete time, k.

Note that when combining two or more artificial neurons we are getting an artificial neural network [47].

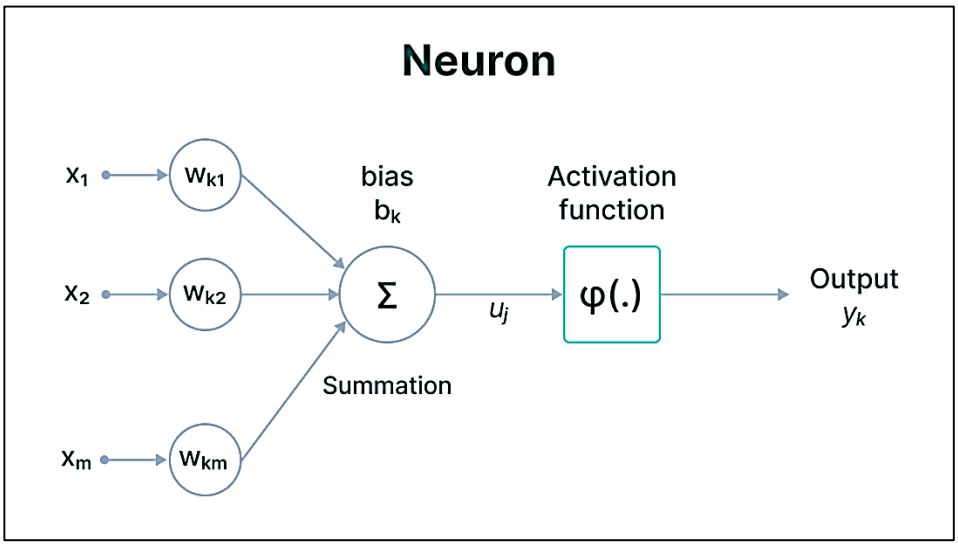


Figure 5 - Neuron in Artificial neural network components

2.3.2. *ANNs architecture*

In the Figure 2.4 [51], it is illustrated that a neural network made of interconnected neurons. Each of them is characterized by its weight, bias, and activation function. Here is a brief explanation of other elements of this network [55].

Input Layer: The input layer takes raw input from the domain. No computation is performed at this layer. Nodes here just pass on the information (features) to the hidden layer.

Hidden Layer: As the name suggests, the nodes of this layer are not exposed. They provide an abstraction to the neural network.

The hidden layer performs all kinds of computation on the features entered through the input layer and transfers the result to the output layer.

Output Layer: It's the final layer of the network that brings the information learned through the hidden layer and delivers the final value as a result.

All hidden layers usually use the same activation function. However, the output layer will typically use a different activation function from the hidden layers. The choice depends on the goal or type of prediction made by the model

2.3.3. *Activation functions*

An activation function decides whether or to what extent the input signal should pass. The node (or neuron) is activated if there is information passing through it. It is a kind of function that maps a real number to a number between 0 and 1 (with rare exceptions).

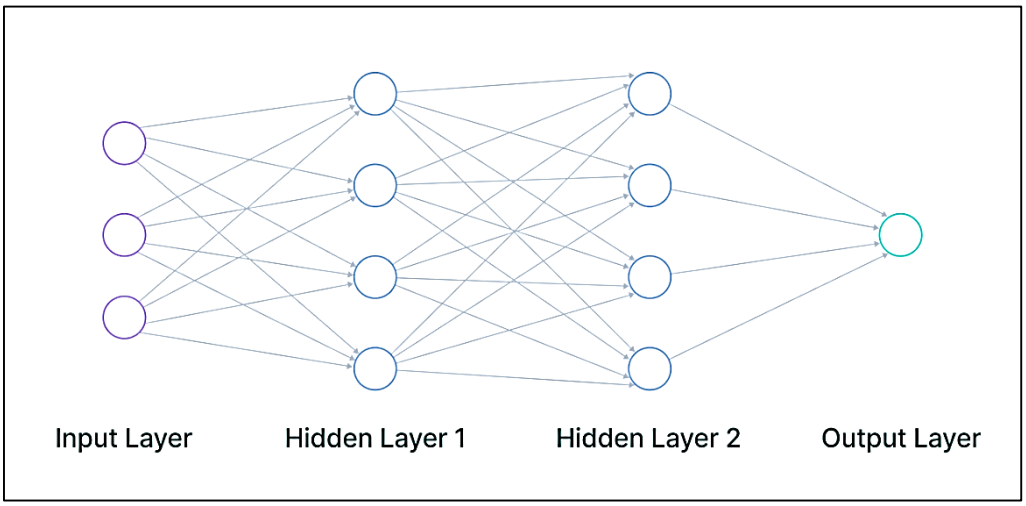


Figure 6 - Neural Networks basic Architecture

which represents the activation of the neuron, where 0 indicates deactivated and 1 indicates fully activated [56], We choose it on the basis of problem that artificial neuron

(artificial neural network) needs to solve [47] .The purpose of activation functions is to add the non-linearity to the neural network.

In the following we introduce some commonly used Activation functions.

- ***Sigmoid or Logistic Activation Function:*** It is the most widely used activation function as it is a non-linear function. Sigmoid function transforms the values in the range 0 to 1 [57]. It can be defined as:

$$f(x) = \frac{1}{e^x}$$

The main reason why we use sigmoid function is because it exists between (0 and 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice [58].

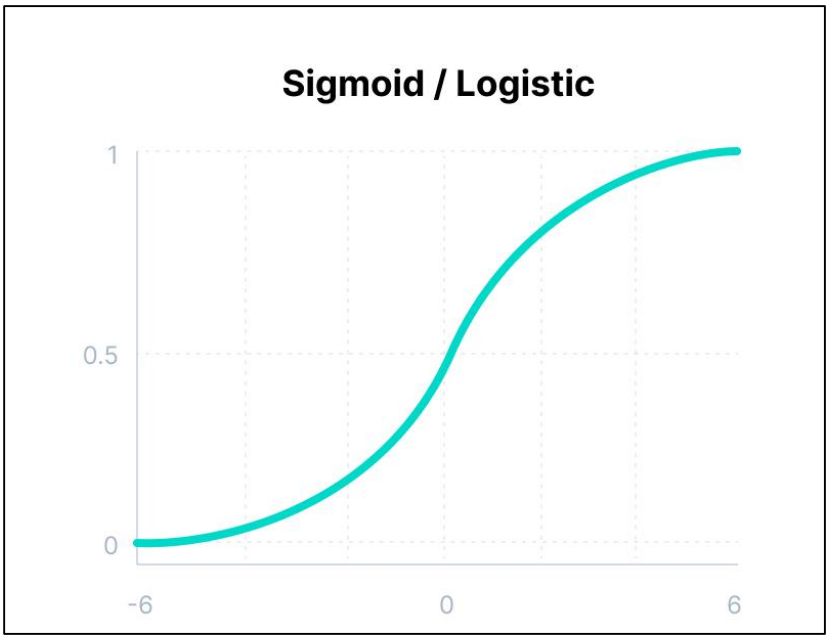
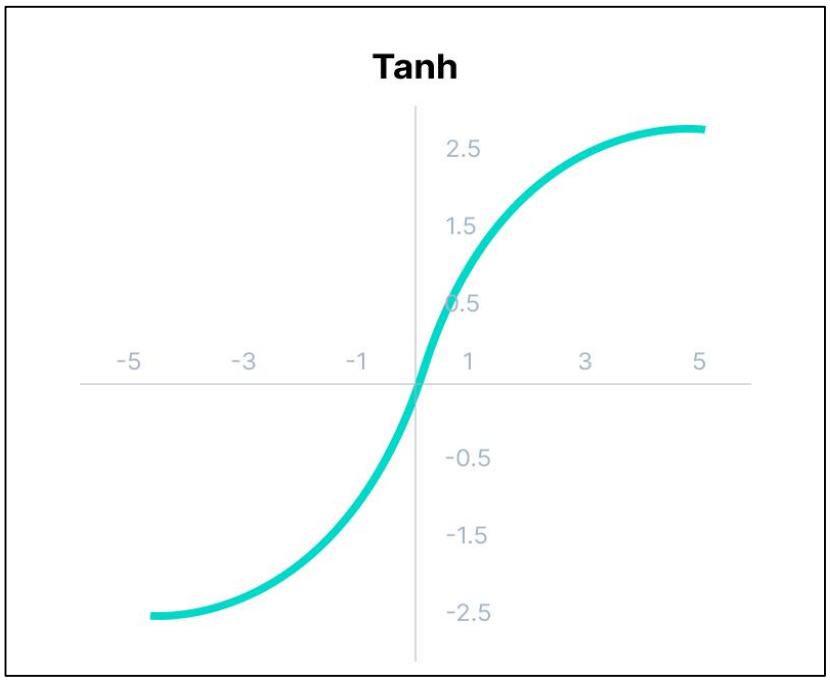


Figure 7 - Sigmoid Activation function

- ***Tanh Function (Hyperbolic Tangent).*** It is Hyperbolic Tangent function. Tanh function is similar to the sigmoid function but it is symmetric to around the origin. This results in different signs of outputs from previous layers which will be fed as input to the next layer [57]. It can be defined as:

$$f(x) = 2sigmoid(2x) - 1$$

Figure 8 - Tanh activation function



- ***ReLU Function.*** ReLU stands for rectified liner unit and is a non-linear activation function which is widely used in neural network. The upper hand of using ReLU function is that all the neurons are not activated at the same time. This implies that a neuron will be deactivated only when the output of linear transformation is zero [57]. It can be defined mathematically as:

$$f(x) = \max(0, x)$$

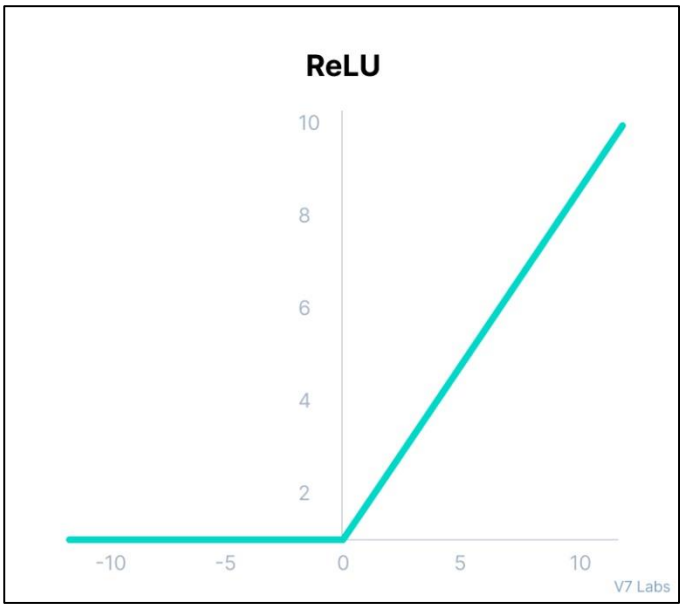


Figure 9 - ReLU activation function

2.3.4. Feed-forward Neural Network

A feed forward neural network (FNN) is a type of artificial neural network where information propagates in a forward direction, without feedback connections. It consists of an input layer that receives input data, one or more hidden layers responsible for processing intermediate representations, and an output layer that produces the final output. Each layer is composed of interconnected nodes, called neurons, which perform computations on the received input using activation functions. The network's weights and biases are adjusted during training to optimize the model's performance [59, 60].

The number of neurons in each layer and the connectivity pattern between layers determine the architecture of the FNN. In a fully connected FNN, each neuron in a given layer is connected to every neuron in the subsequent layer. This allows the flow of information throughout the network, enabling the network to learn complex relationships within the data [59].

The hidden layers serve as information-processing stages, where each neuron applies a non-linear activation function to its input. Common activation functions used in FNNs include the sigmoid function, ReLU (Rectified Linear Unit), and tanh (hyperbolic tangent) function. These activation functions introduce non-linearity into the network, enabling it to model complex relationships and capture non-linear patterns in the data [60].

In order to propagate the features representation to the next layer (forward pass), we perform the equation below [61]:

$$H^{[i+1]} = \sigma(W^{[i]}H^{[i]} + b^{i})$$

Equation 1 Forward Pass in Neural Networks

Where:

- H the feature representation at layers
- i the number of layers
- σ the activation function
- W the weight
- b *the* bias

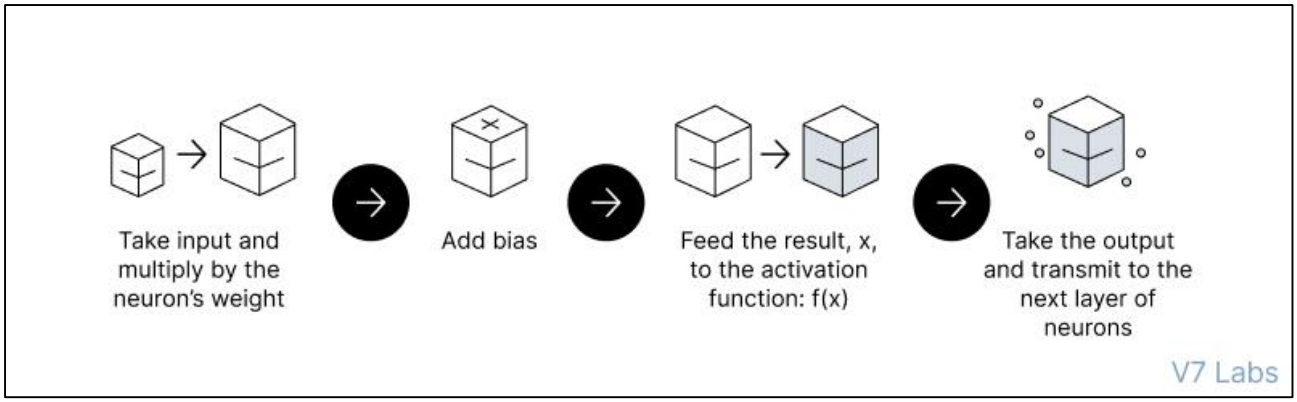


Figure 10 - Forward propagation in neural networks

2.4. Recurrent Neural Networks

2.4.1. Overview

Recurrent Neural Networks (RNNs) are a type of artificial neural network specifically designed to process sequential data by introducing the concept of recurrent connections. Unlike feed forward neural networks, RNNs have connections that form a directed cycle, allowing them to retain and utilize information from previous time steps or inputs [60, 62].

RNNs process sequential data by iteratively applying the same set of weights and biases across multiple time steps. At each time step, the network takes an input, computes the hidden state based on the previous hidden state and current input, and produces an output. This recurrent computation enables RNNs to model dynamic temporal relationships in sequential data, making them suitable for tasks such as speech recognition, language modelling, machine translation, and time series analysis [60, 62].

However, standard RNNs suffer from the vanishing or exploding gradient problem, where the influence of past inputs on the current hidden state diminishes or amplifies exponentially over time. To address this issue, various advanced RNN architectures have been developed, such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs). These architectures incorporate gating mechanisms that allow the network to selectively retain or forget information, improving its ability to learn and capture long-term dependencies in sequential data [63, 64]

2.4.2. RNNs Architecture

As shown in Figure 2.7 [65] the architecture of a Recurrent Neural Network (RNN) consists of input, hidden, and output layers, with a feedback loop that enables the network to process sequential data by maintaining hidden states capturing contextual information from previous inputs [60, 62]. In the following section we will discuss them briefly.

- The input layer of an RNN represents the initial input to the network. Each element of the input sequence is represented as a vector, and the sequence is fed into the network step by step. This step is similar to the input layer of other neural network architectures [60].
- The hidden layer of an RNN is responsible for maintaining and updating the hidden state, which captures information from previous inputs. It takes both the current input and the previous hidden state as input and produces a new hidden state. This recurrent connection allows the network to retain memory of past inputs and capture sequential dependencies [62].
- In the Output Layer of an RNN takes the hidden state as input and produces the output for the current step. The output can be a prediction, classification, or any

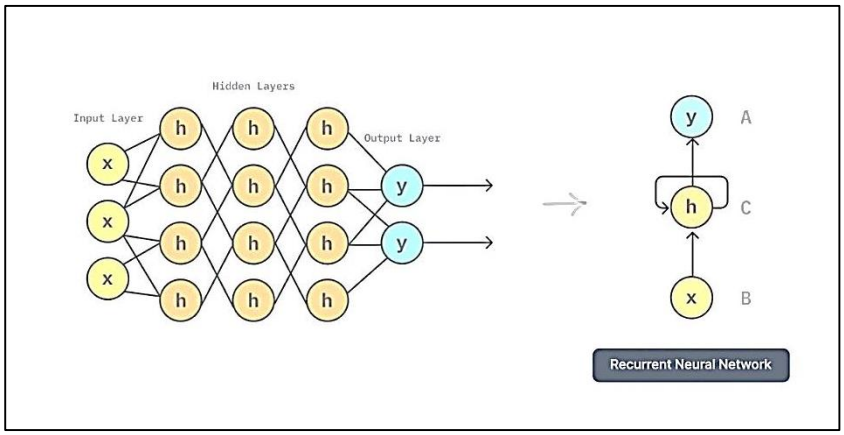


Figure 11 - a simple RNNs architecture

other desired output based on the task at hand. The output layer can be a fully connected layer or any other appropriate layer for the specific task [66].

- The next step consists of the feedback loop which allows the output from the current step to be fed back into the network as the input for the next step. This feedback mechanism is crucial for capturing sequential dependencies and maintaining memory of past inputs. It enables the RNN to process the entire input sequence iteratively [67].
- Finally, the Time Unfolding step to process the entire input sequence, the RNN is time-unfolded into multiple steps, where each step represents a single time step of the input sequence. This unfolding allows the RNN to process the sequential data

one step at a time and facilitates the application of back propagation through time (BPTT) for training the network [67].

2.5. Convolutional neural networks

2.5.1. Overview

Convolutional neural networks (CNNs) are special versions of FNNs. FNNs are usually fully connected networks while CNNs preserve the local connectivity.as shown in figure 2.7 The CNN architecture usually contains convolutional layers, pooling layers, and several fully connected layers or dense layers [56].

2.5.2. CNN Architecture

As shown in the Figure 2.7 [68] The Convolutional layer applies filters to the input image to extract features, the Pooling layer down samples the image to reduce computation, and the fully connected layer makes the final prediction [69], they also consist of neurons that have trainable weights and bias. Each neuron receives and transforms some information from previous layers. The difference is that some of the neurons in CNNs may have different designs from the ones we introduced for feed forward networks [53], we will provide a brief explanation of every step.

According to [70, 71] :

- The input to a CNN is usually an image or a set of images. An image is represented as a grid of pixels, where each pixel contains colour information (RGB values). The dimensions of the input image are typically fixed and known in advance.
- While the convolutional layer is the primary building block of a CNN. It applies a set of learnable filters (also known as kernels) to the input image to extract features. Each filter performs a convolution operation by sliding over the input image and computing dot products between the filter weights and local regions of the input. The output of this operation is called a feature map or an activation map.

The convolutional layer helps the network learn spatial hierarchies of features by capturing low-level features in the earlier layers and more complex features in the deeper layers.

After the convolution operation, an element-wise activation function is applied to introduce non-linearity into the network. Commonly used activation functions include Rectified Linear Units (ReLU), sigmoid, and hyperbolic tangent (tanh) [72].

- Then the pooling layer reduces the spatial dimensions of the feature maps while retaining the most important information. It works by partitioning the input feature map into small regions (e.g., 2x2 or 3x3) and taking the maximum or average value within each region. Pooling helps to achieve translation invariance and reduces the sensitivity of the network to small spatial variations in the input [73, 74].

- Finally, the dense layer connects every neuron from the previous layer to every neuron in the current layer. It transforms the features learned by the convolutional layers into a vector of class scores or probabilities. This layer is typically placed at the end of the network and provides the final classification or regression output [75].

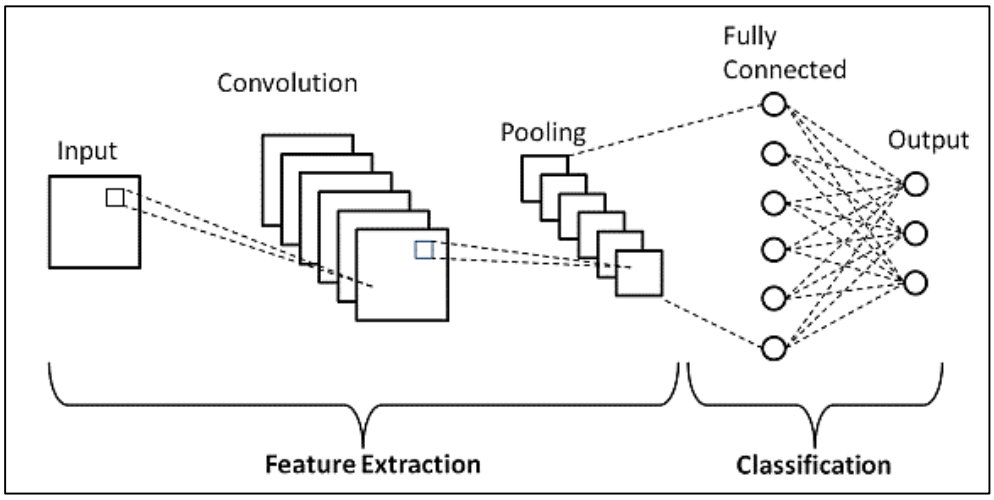


Figure 12 - a simple convolutional neural networks architecture

2.6. Graph fundamentals

2.6.1. Overview

Graphs are ubiquitous structures that can be used to model complex relationships between entities. Graphs have been used in many fields such as social network analysis, recommendation systems, computer networks, and bioinformatics. In recent years, deep learning techniques have been applied to graphs to extract features and learn representations, resulting in Graph Neural Networks (GNNs) that have shown promising results in various applications, including session-based recommender systems.

In this section, we will provide an introduction to graphs and their importance in machine learning. We will discuss the different types of graphs, graph algorithms, and graph representations. We will also briefly introduce the concept of GNNs and their application in session-based recommender systems.

2.6.2. Graph representation

In this section, we introduce the definition of graphs. With brief explanations:

Definition 1: A graph is often denoted by $G = (V, E)$, where V is the set of vertices and E is the set of edges. An edge $e = u,v$ has two endpoints u and v, which are said to be joined by e. In this case, u is called a neighbour of v, or in other words, these two vertices are adjacent. Note that an edge can either be directed or undirected. A graph is called a directed graph if all edges are directed or undirected graph if all edges are undirected. The degree of vertices v, denoted by $d(v)$, is the number of edges connected with v [56]. The figure 13 illustrate directed and undirected graph [76] .

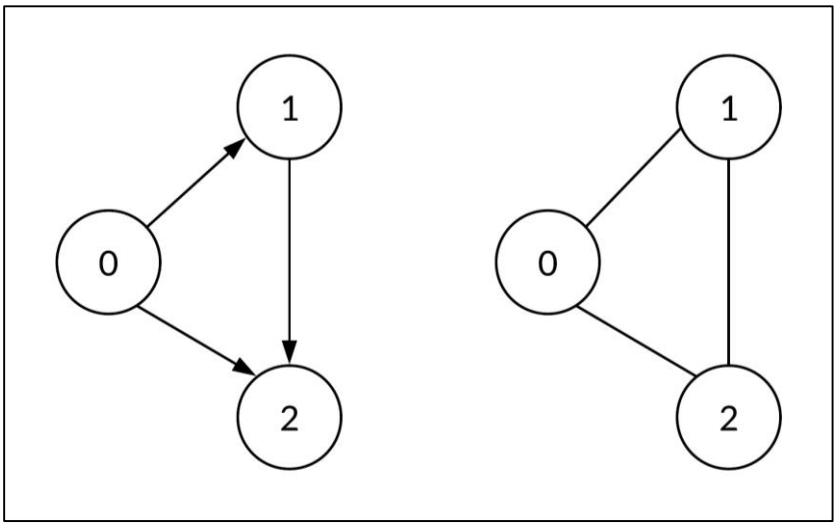


Figure 13 - Directed and undirected graph

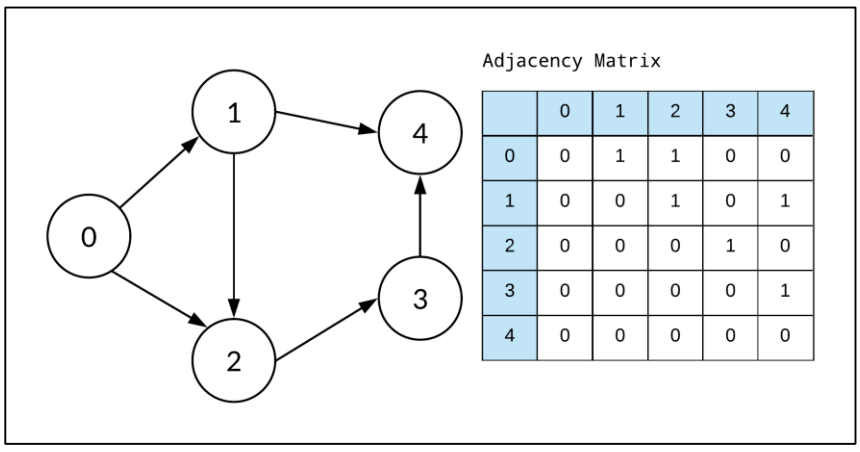
2.6.3. Algebra representation of graphs

Here are a few helpful algebraic representations for graphs:

- ***Adjacency matrix:*** For a simple graph with vertex set $U = \{u_1, \ldots, u_n\}$, the

  Adjacency matrix is a square $n \times n$ matrix A such that its element A_{ij} is one when there is an edge from vertex u_i to vertex u_j, and zero when there is no edge [77], the Figure represent an example of adjacency matrix [78] :

- ***Degree matrix:*** Given a graph $G=(V,E)$ with $iVi = n$, the degree matrix D for G in



	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

Figure 14 - Adjacency matrix of a graph

an $n \times n$ diagonal matrix defined as [79]

$$D_{i,j} := \begin{cases} deg(v_i) & if\ i = j \\ 0 & otherwise \end{cases}$$

where the degree $\boldsymbol{deg}(\boldsymbol{v_i})$ of a vertex counts the number of times an edge terminates at that vertex. In an undirected graph, this means that each loop increases the degree of a vertex by two. In a directed graph, the term *degree* may refer either to in degree (the number of incoming edges at each vertex) or out degree (the number of outgoing edges at each vertex).

- ***Laplacian matrix***: Given a simple graph G with n vertices $v_1, \ldots, v_n$, its Laplacian matrix $L_{n\times n}$ is defined element-wise as

$$L_{n\times n} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Or equivalently by the matrix

$L = D - A$

Where: D is the degree matrix and A is the adjacency matrix of the graph. Since G is a simple graph, A only contains 1s or 0s and its diagonal elements are all 0s.

2.6.4. Computational Tasks on Graphs

We've discussed graph examples in the wild, but what tasks can we perform on this data? There are three types of prediction tasks on graphs: node-level, graph-level, and edge-level tasks.

- **Node-focused tasks:** Node-focused tasks involve analysing and understanding individual nodes in a graph. These tasks aim to extract meaningful information from each node, such as its properties, attributes, or importance within the graph. Examples of node-focused tasks include node classification, node clustering, and node ranking. Node classification aims to assign a category or label to each node based on its properties. Node clustering involves grouping similar nodes together. Node ranking determines the importance or centrality of each node in the graph [80, 81].
- **Graph-focused tasks:** Graph-focused tasks involve analysing the entire graph structure as a whole. These tasks aim to understand the overall properties, characteristics, and patterns present in the graph. Examples of graph-focused tasks include graph classification, graph generation, and graph similarity. Graph classification involves assigning a category or label to an entire graph based on its structure or properties. Graph generation focuses on generating new graphs that exhibit similar properties or characteristics as the input graph. Graph similarity aims to quantify the similarity between two or more graphs [82, 83].
- **Edge-level tasks:** Edge-level tasks involve analysing the connections or relationships between pairs of nodes in a graph. These tasks focus on understanding the interactions, dependencies, or patterns that exist between the nodes through their connecting edges. Examples of edge-level tasks include link prediction, graph matching, and graph alignment. Link prediction aims to predict missing or future edges in a graph based on the existing connections. Graph matching involves finding

correspondences between nodes in two or more graphs. Graph alignment aims to align or map nodes between different graphs based on their structural similarities [84, 85].

2.6.5. Graph applications

Graphs can be used to model many types of relations and processes in physical, biological [86, 87], social and information systems [88]. Graphs can effectively depict numerous practical issues. By placing a strong focus on their relevance to real-life systems, the notion of a network can be defined as a graph where attributes (such as names) are linked to the vertices and edges. The field those studies and comprehends real-world systems as networks is known as network science. The following sections discuss several real-world applications of graphs:

- ***Computer science:*** Graphs are used to model many problems and solutions in computer science, such as representing networks, web pages, and social media connections. Graph algorithms are used in path finding, data compression, and scheduling or it might be used in Artificial intelligence to model and analyse data in many AI applications, such as machine learning, Artificial Intelligence, and natural language processing [89].
- **Citation networks as graphs:** Scientists routinely cite other scientists' work when publishing papers. We can visualize these networks of citations as a graph, where each paper is a node, and each *directed* edge is a citation between one paper and another. Additionally, we can add information about each paper into each node, such as a word embedding of the abstract.
- ***Social science:*** Graph theory is also widely used in sociology as a way, for example, to explore rumour spreading, notably through the use of social network analysis software [90]. Under the umbrella of social networks are many different types of graphs [91].
- ***Mathematics:*** In mathematics, graphs are useful in geometry and certain parts of topology such as knot theory. Algebraic graph theory has close links with group theory. Algebraic graph theory has been applied to many areas including dynamic systems and complexity [90].

2.7. Graph Neural Networks

2.7.1. Overview

Graph Neural Networks are a subclass of Deep Learning techniques that are specifically built to do inference on graph-based data. They are applied to graphs and are capable of performing prediction tasks at the node, edge, and graph levels [92]. GNN is a neural network that is directly applied to graphs, giving a handy method for performing edge, node, and graph level prediction tasks.

2.7.2. GNNs architecture

GNN architecture's primary goal is to learn embedding that contains information about its neighbourhood. We may use this embedding to tackle a variety of issues, including node labelling, node and edge prediction, and so on [92].

Figure 2.8 shows the general architecture of a graph neural network where the input graph is fed into the hidden nodes to learn the representations of graph-structured data, and the output graph is generated from the learned graph-structured representations. It works by propagating information along the edges of a graph to update the node representations. In other words, the network iteratively aggregates the information from a node's neighbouring nodes and uses this information to update its own representation. This process is repeated for multiple iterations until the nodes converge to a stable representation.

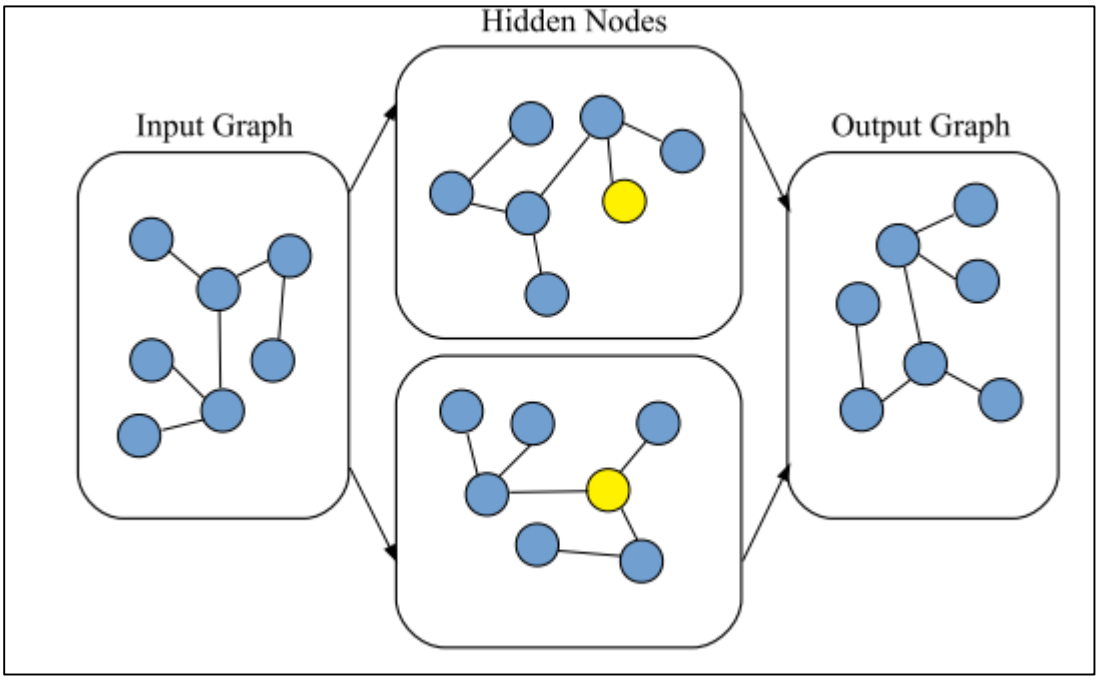


Figure 15 - Graph Neural Network architecture

2.7.3. GNNs types

Graph neural networks are classified into three types [92] :

- ***Recurrent Graph Neural Network:*** Consider as the most powerful GNN variant, also known as RecGNN
- ***Spatial Convolutional Network:*** Spatial Convolutional Networks have a similar idea to CNNs. As is well known in CNN, convolution is performed by summing the neighbouring pixels around a central pixel using a filter and learnable weights. Spatial Convolutional Networks operate on a similar principle, aggregating the properties of neighbouring nodes toward the centre node.

- ***Spectral Convolutional Network:*** In comparison to other types of Graph Neural Networks, this sort of GNN is built on a solid mathematical foundation. It is based on the theory of Graph Signal Processing. It simplifies by the use of Chebyshev polynomial approximation.

2.7.4. GNNs tasks

According to [93] , Graph neural networks tasks can be described as follow:

- ***Graph Classification:*** We use this to classify graphs into various categories. Its applications are social network analysis and text classification.
- ***Node Classification:*** this task uses neighbouring node labels to predict missing node labels in a graph.
- ***Link Prediction:*** predicts the link between a pair of nodes in a graph with an incomplete adjacency matrix. It is commonly used for social networks.

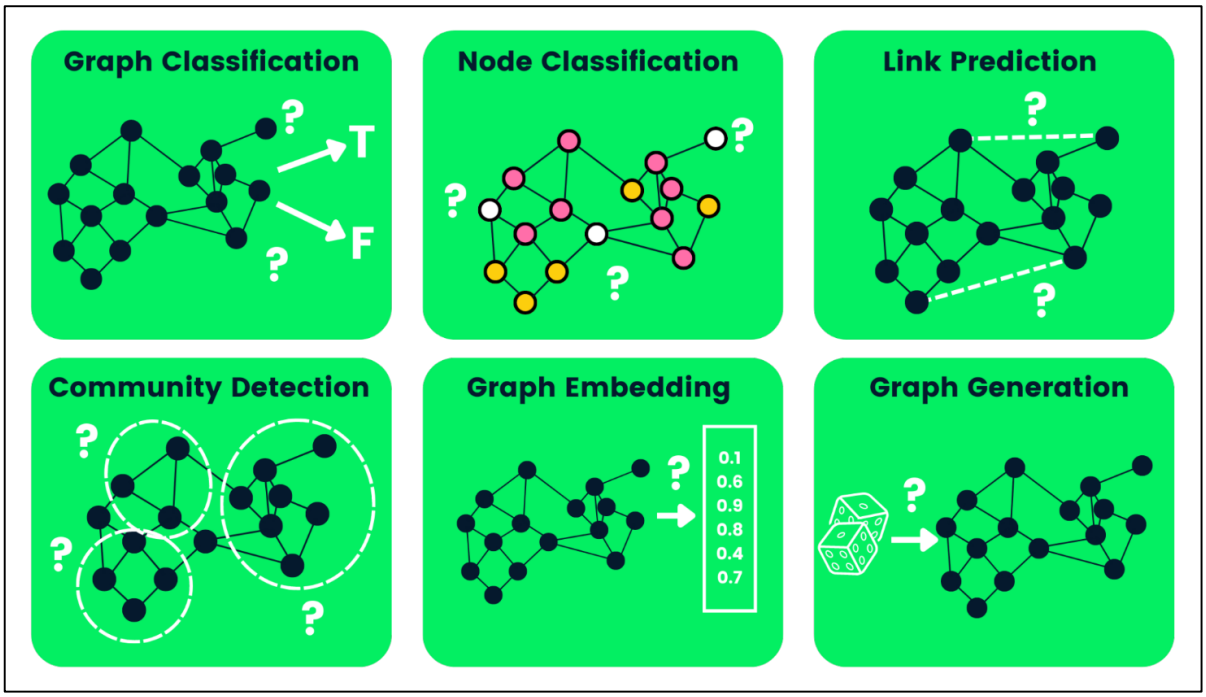


Figure 16 - Graph Neural Networks tasks

- ***Community Detection***: Divides nodes into various clusters based on edge structure. It learns from edge weights and distance and graph objects similarly.
- ***Graph Embedding****:* Maps graphs into vectors, preserving the relevant information on nodes, edges, and structure.
- ***Graph Generation****:* Learns from sample graph distribution to generate a new but similar graph structure.

2.7.5. GNNs advantages and limitations

The main advantage of using graph neural networks is their ability to handle complex graph-structured data. Additionally, GNNs can be used for both supervised and unsupervised learning tasks, making them a versatile tool for many applications [92].

According to [92] graph neural networks have one limitation is that they can be computationally expensive, especially for large graphs. Additionally, GNNs can suffer from over fitting, especially when the graph structure is noisy or incomplete. Finally, the interpretability of GNNs can be a challenge, as it is often difficult to understand how the network arrives at its predictions.

2.8. Graph Convolutional Networks

2.8.1. Main ideas

As the name "Convolutional" suggests, the idea was from Images and then brought to Graphs. However, when Images have a fixed structure, Graphs are much more complex.

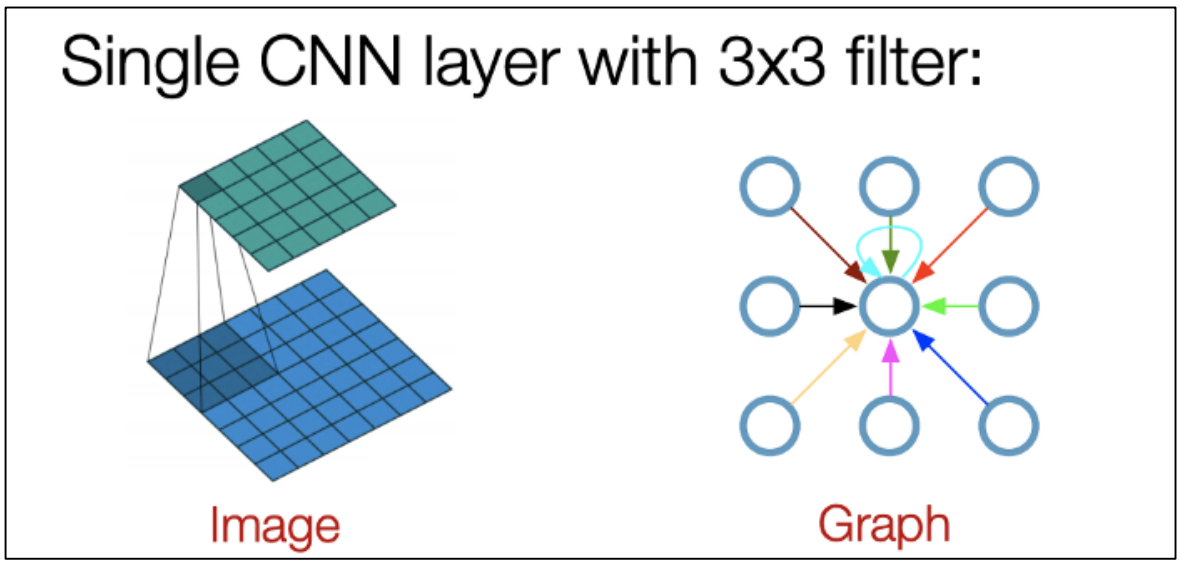


Figure 17 - Convolution idea from images to graphs

2.8.2. Definition and principals

Graph Convolutional Networks (GCNs) are a class of deep learning models designed to operate on graph-structured data. Unlike traditional neural networks that process grid-like data, GCNs extend neural network architectures to handle non-Euclidean domains represented by graphs or networks [94, 95], The core idea behind GCNs is to generalize the concept of convolutional layers from grid-like data (such as images) to graphs. In traditional convolutional neural networks (CNNs), convolutions operate on local neighbourhoods of pixels, exploiting the grid structure. In GCNs, convolutions are defined in the spectral or spatial domain of graphs, leveraging the connectivity patterns between nodes [94, 95] , GCNs typically operate in a message-passing framework, where each node receives and aggregates information from its neighbouring nodes. This aggregation process

is analogous to the receptive field in CNNs, allowing nodes to gather information from their local graph neighbourhood. The gathered information is then used to update the node's representation or features. By iteratively propagating and aggregating information across the graph, GCNs learn to capture the graph structure and perform node-level or graph-level predictions [94, 95] .

2.8.3. GCN architecture

As the Figure 18 [61] show an illustration of Graph convolutional networks we can consider that GCN is similar to the filter in convolution and denotes the definition of convolution from the regular grid to irregular structures like graphs [96]. The node embedding are updated as follows [97]:

$$H^{[l+1]} = \delta\left(D^{-1/2}AD^{-1/2}H^{l}w^{l}\right)$$

Where H^{l} represent the embedding matrix of l - th layer convolution, d is the embedding dimension. A is the adjacency matrix with self-loops, and $D_{ii} = \sum_j A_{ij}$ is the degree matrix. All these operations are making sure that all the neighbours are receiving an aggregated message from multiple hop neighbours. The weights W^{l} in the GCN are trained using gradient descent.

GCNs are the simplified version of Graph Convolutional Neural Networks (GCNNs). A typical GCN consist of three steps

1) Feature propagation,

2) Linear transformation,

3) Application of a non-linear activation function [98].

Feature propagation is achieved using convolutional matrix computed from graph topology. For the linear transformation different parameters are learned to minimize a loss function and for that typical activation functions such as sigmoid or ReLU are used [97].

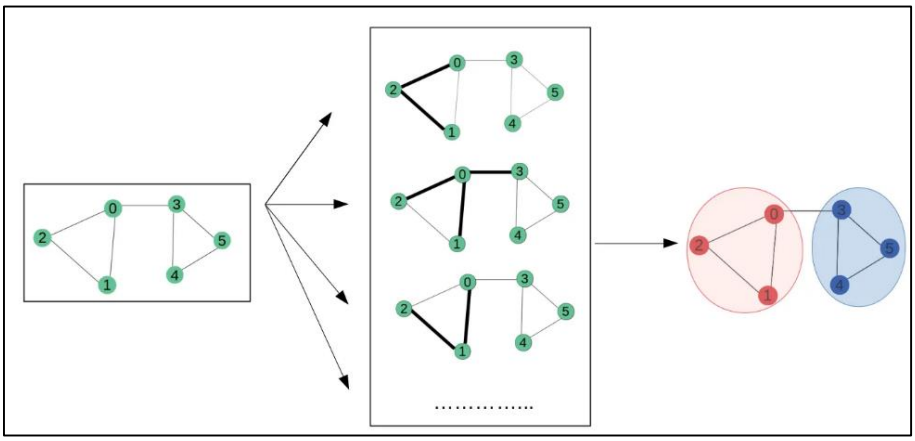


Figure 18 - Illustration of Graph Convolutional Networks

Let's explain the GCNs steps more deeply:

Convolution and message passing (feature propagation): In the feature propagation step, GCNs aim to capture and propagate information across the graph by considering the features of neighbouring nodes. This is achieved by computing a weighted sum of the features of each node's neighbours and incorporating it into the node's own feature representation. The weights are typically determined based on the graph structure or learned through a training process. This aggregation of neighbouring features helps to capture the local context and dependencies in the graph [99].

As shown in the Figure 19 [100] , for each node, we get the feature information from all its neighbours and of course, the feature of itself. Assume we use the *average ()* function. We will do the same for all the nodes. Finally, we feed these average values into a neural network.

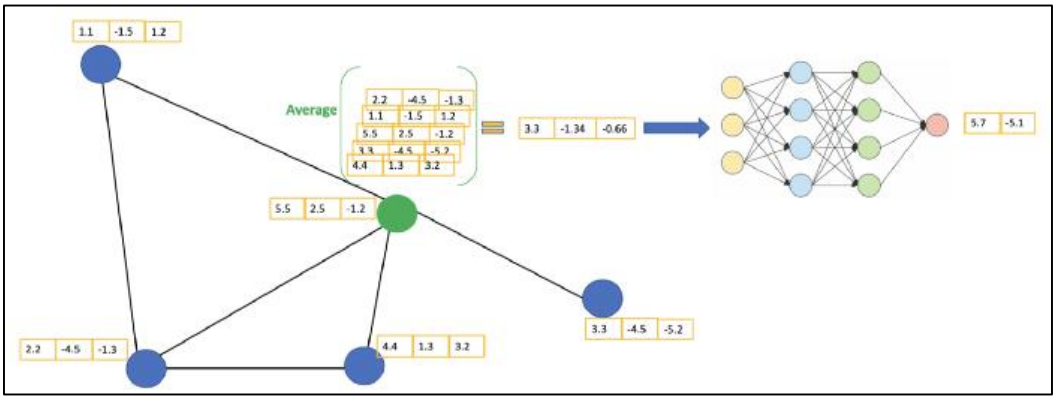


Figure 19 - convolution method in GCN

Note that the main idea of GCN. Consider the green node (see Figure 19). First, we take the average of all its neighbours, including itself. After that, the average value is passed through a neural network. Note that, in GCN, we simply use a fully connected layer. In this example, we get 2-dimension vectors as the output (2 nodes at the fully connected layer).

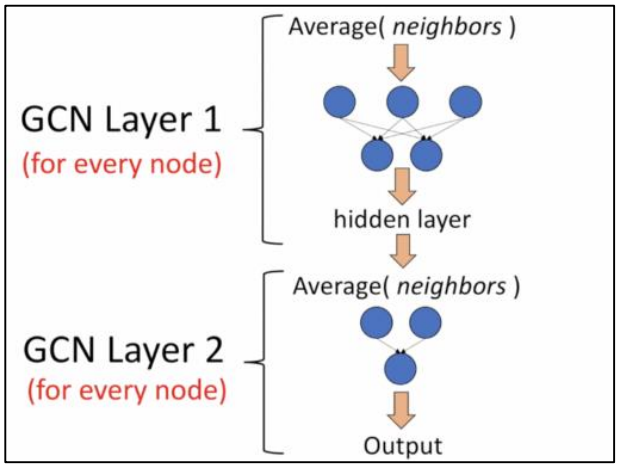


Figure 20 - GCN with two layers

An important note to consider is shown in Figure 20 [100] that illustrates an example of 2-layer GCN: The output of the first layer is the input of the second layer. Again, note that the neural network in GCN is simply a fully connected layer.

- ***Linear transformation:*** After the feature propagation step, a linear transformation is applied to the updated node features. This transformation aims to learn a more expressive and task-specific representation by mapping the aggregated features to a new feature space. The linear transformation is typically implemented as a matrix multiplication between the updated features and a learnable weight matrix. The weight matrix captures the relationships between the input features and the desired output [95].

- ***Application of a Non-linear Activation Function:*** To introduce non-linearity and capture more complex patterns, an activation function is applied element-wise to the transformed features. This non-linear mapping allows the GCN to model non-linear relationships between features and enables the network to learn more expressive representations. Common activation functions include the rectified linear unit (ReLU), sigmoid, or hyperbolic tangent (tanh) [101].

2.8.4. GCN variations

In the field of Graph Convolutional Networks (GCNs), there are several methods and variations that have been developed to enhance the performance and address specific challenges in graph-based learning tasks. In this section we will discuss some of them.

- ***GraphSage.*** GraphSage (Graph Sample and Aggregated) is a scalable GCN method that performs inductive learning on large-scale graphs. It aggregates information from a node's local neighbourhood by sampling and aggregating features, allowing generalization to unseen nodes during inference [99].
- ***Graph Attention Network (GAT).*** GAT introduces an attention mechanism into GCNs, allowing nodes to assign different importance weights to their neighbours during the aggregation step. Attention mechanisms enable the network to focus on more relevant information, enhancing the representation learning process [102].

- ***ChebNet***. ChebNet utilizes Chebyshev polynomial filters to approximate the spectral filters in GCNs. This approach enables efficient graph convolution by avoiding the computationally expensive Eigen decomposition step, making it scalable for large graphs [103].
- ***Graph Isomorphism Network (GIN).*** GIN is a variant of GCN that operates by aggregating information from the neighbouring nodes and updating node features using a sum-aggregation function. It applies multiple graph isomorphism network layers to capture higher-order structural information in the graph [104].

These are just a few notables' variants of GCNs. The field of graph representation learning is evolving rapidly, and new variants and improvements are continuously being developed.

2.8.5. GCN types

GCNs mainly include two categories: spectral-based GCNs and spatial-based GCNs [105]

- ***Spectral-based GCNs.*** The first notable spectral-based graph convolutional network is proposed by Bruna et al [106], it operate in the spectral domain by leveraging the graph Laplacian eigen basis. They utilize the graph Fourier transform to transform the graph signals into the spectral domain, where convolutions are applied. The graph Laplacian Eigen basis captures the global structure of the graph. Spectral-based GCNs utilize the graph Laplacian eigenvalues and eigenvectors to define the graph convolution operation [103, 107]

It can be defined as fellow:

$$Y = \sum_{k=0}^{K} U \Lambda^k U^T X W_k$$

Where:

Y is the output feature matrix,

X is the input feature matrix,

W represents the trainable weight matrix for the *k-th* graph convolutional layer,

U and Lambda denote the eigenvectors and eigenvalues of the graph Laplacian, respectively,

K is the number of layers.

- ***Spatial-based GCNs***. Spatial-based GCNs, also known as neighbourhood aggregation or message-passing GCNs operate in the spatial domain by aggregating information from the local neighbourhood of each node. They propagate information through message passing between neighbouring nodes, which captures the local structure and relationships in the graph. Spatial-based GCNs update the node features by aggregating and transforming the features of neighbouring nodes [94, 99]

Its operation can be defined as fellow:

$$Y = \sigma(D^{-1}AXW)$$

Where:

Y is the output feature matrix,

X is the input feature matrix,

W represents the trainable weight matrix,

A denotes the adjacency matrix of the graph,

D is the degree matrix, which is a diagonal matrix with the node degrees as its diagonal entries,

σ Represents the activation function.

2.8.6. An explanation example

We will provide a simple example to explain how GCN works according to the steps mentioned earlier.

Consider a social network graph where nodes represent individuals and edges represent their connections. Each node has associated features such as age, gender, and interests. The task at hand is to predict whether two individuals are likely to be friends based on their shared interests.

Feature Propagation: During feature propagation, a GCN would aggregate the features of each node's neighbours to update its own feature representation. In the social network context, this could involve summing or averaging the features of a node's immediate friends to capture the common characteristics within their social circles. For instance, if Alice and Bob share similar interests and have several mutual friends, their features would be updated to reflect this shared information.

Linear Transformation: After the feature propagation step, a linear transformation is applied to the updated node features. The linear transformation aims to learn a weighted combination of the features that is relevant for the prediction task. In our example, the weights would capture the importance of different features (e.g., age, gender, interests) in determining the likelihood of friendship. By applying this transformation, the GCN can extract more discriminative representations from the updated features.

Application of a Non-linear Activation Function: To introduce non-linearity, an activation function is applied to the transformed features. This activation function can capture complex relationships and patterns within the data. For instance, it can model that individuals with similar interests and close ages are more likely to form friendships. By applying the activation function, the GCN can learn and represent these non-linear relationships, allowing for more accurate predictions.

By combining these steps, a GCN can effectively leverage the graph structure and the shared information between nodes to predict friendships in the social network.

2.9. Conclusion

In conclusion, this chapter provided an extensive overview of key concepts and techniques in graph analysis and neural networks. We began by introducing the fundamentals of graphs; Next, we delved into the field of machine learning, discussing the three main learning processes: supervised learning, unsupervised learning, and reinforcement learning. We emphasized the importance of artificial neural networks (ANNs) and their architecture, activation functions, and explain their types (FNNs, RNNs, CNNs, and finally GNNs) briefly, note that a major focus of this chapter was on graph neural networks (GNNs), which are specifically designed to handle graph-structured data.

Furthermore, we introduced graph convolutional networks (GCNs), a type of GNN that utilizes graph convolution operations.

Overall, this chapter serves as a solid foundation for the subsequent chapter of this thesis, as it has covered fundamental concepts in graph analysis and provided insights into the neural network architectures that are specifically designed for graph data. The knowledge gained here will be crucial for design and implement a session-based recommender system using GCNs in the following chapter.

Chapter 3.

Graph Convolutional Networks for the Development of Session-based Recommender Systems

3.1. Introduction

In this chapter, we will present our algorithm and its underlying principles for session-based recommendation. We will then compare its performance with existing works in the field, specifically those using graph neural networks. Finally, we will discuss the results obtained from our experiments, providing insights into the strengths and limitations of our approach.

3.2. Presentation of our GCN-based Model

3.2.1. *Main idea*

Our model proceeds in three main phases: session presentation phase, GCN Model phase, and Recommendation phase.

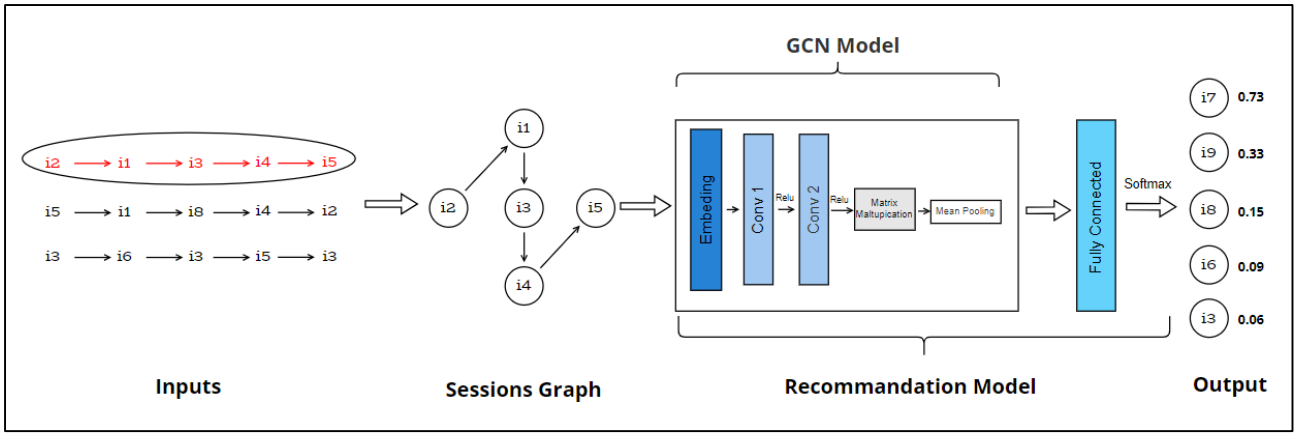


Figure 21 - Overview on GCN-Based SBRSs Model

3.2.2. *Session representation*

Each session sequence s can be modelled with the adjacency matrix as a directed graph G = (I ; E). In this session graph, each node represents an item i and each edge means that a user clicks item i1 after i2 in the session.

3.2.3. *GCN model*

It consists of an embedding layer, followed by two convolutional layers with ReLU activation functions. The key layers and operations in the model are:

- Embedding Layer: The self-Embedding layer maps item indices to dense vectors of hidden dimensions. It learns meaningful representations for the items in the graph.
- Convolutional Layers: The model has two convolutional layers: self.conv1 and self.conv2.These layers perform 1-dimensional convolutions on the input data. Each convolution is followed by a Rectified Linear Unit (ReLU) activation function, applied using F.relu. The convolutions help capture the local relationships and patterns in the data.
- Message Aggregation: After the convolutions, the model performs message aggregation from neighbouring nodes. The adjacency matrix is multiplied with the

output of the convolutions using torch.matmul (adj, x). This operation combines information from connected nodes to enrich the representation of each item.

- Mean Pooling: The model uses mean pooling to aggregate the messages from neighbors. The output of the message aggregation step is permuted, and then the mean is taken along the second dimension using x.mean (dim=1). This results in a single representation for each item in the graph.

3.2.4. Recommendation step

The SessionRec model combines the graph convolutional capabilities of the GCN model with a fully connected layer and log-SoftMax activation to generate recommendations.

The fully connected layer: also known as the linear layer or dense layer is a fundamental component in neural networks. It performs a linear transformation on the input data, mapping it to a different dimensional space.

In the context of the provided code, the fully connected layer self.fc takes the item representations generated by the GCN component as input. It applies a linear transformation to these representations, mapping them to the number of output items.

The purpose of the fully connected layer is to learn and capture complex relationships between the input data and the desired output. It introduces non-linearity and helps the model make more complex predictions by combining and weighting the input features.

Mathematically, the fully connected layer computes the following operation:

$$\text{Output} = \text{input} * \text{weights} + \text{bias}$$

Here, input represents the item representations from the GCN component, weights are learnable parameters that the model optimizes during training, and bias is an additional learnable parameter. The output of the fully connected layer is a tensor that undergoes further processing, such as activation functions, to produce the final predictions or features for subsequent layers.

In summary, the fully connected layer adds flexibility and non-linearity to the model, allowing it to learn and capture complex patterns in the data. It plays a crucial role in transforming the intermediate representations from earlier layers into meaningful predictions or features.

The softmax function: is applied after the fully connected layer in the provided code. Its role is to convert the output of the fully connected layer into a probability distribution over the different output classes.

In the context of the recommendation system, the softmax function is used to determine the likelihood or probability of each item being the next recommended item. It assigns higher probabilities to items that are more likely to be relevant or preferred by the user based on the given input.

3.3. Implementation

The implementation was written in Python 3.x versions with Google-Collab IDE and it should work with any Python 3 version, such as Python 3.6, 3.7, 3.8, or 3.9. It utilizes several libraries for different functionalities. Here's an overview of the languages and libraries used in the implementation:

3.3.1. Python. The entire code is written in Python, a popular programming language for data analysis and machine learning.

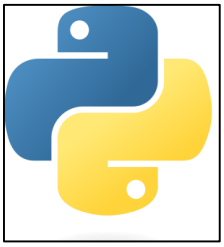


Figure 22 - python logo

Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability and uses indentation as a key part of its syntax, making it easy to write and understand[5].

Key features of Python include:

- ***Easy-to-read syntax:*** Python uses a clean and straightforward syntax, making it easier for beginners to learn and understand. It uses whitespace indentation to define code blocks, eliminating the need for braces or other delimiters.
- ***Versatility:*** Python is a versatile language that can be used for various purposes, including web development, data analysis, machine learning, scientific computing, automation, and more. It has a large standard library and a vast ecosystem of third-party packages that extend its capabilities.
- ***Cross-platform compatibility:*** Python is available on multiple platforms, including Windows, macOS, Linux, and various other operating systems. This allows developers to write code once and run it on different platforms without significant modifications.
- ***Large community and ecosystem:*** Python has a thriving and supportive community of developers worldwide. This community contributes to the development of libraries, frameworks, and tools that make Python suitable for a wide range of applications. Some popular libraries and frameworks in the Python ecosystem include NumPy, Pandas, Django, Flask, TensorFlow, and PyTorch.
- ***Interpretation and scripting:*** Python is an interpreted language, meaning that code is executed line by line at runtime without the need for compilation. This allows

[5] https://www.python.org/

for rapid development and prototyping, as well as easy integration with other languages and systems.

3.3.2. Libraries

- ***PyTorch:*** It is a deep learning framework used for building and training neural networks. The code leverages PyTorch[6] to define and train the session-based recommendation model.
- ***Pandas:*** It is a data manipulation and analysis library. Pandas[7] are used to load and preprocess the MovieLens 100k dataset, as well as perform various data operations such as sorting, grouping, and filtering.
- ***Scikit-learn:*** It is a machine learning library that provides various tools for model selection and evaluation. In the code, scikit-learn[8] is used to split the dataset into train and test sets using the train_test_split function.
- ***NumPy:*** It is a fundamental library for numerical computing in Python. The code uses NumPy[9] to create and manipulate arrays for storing the adjacency matrix and session lengths.

These are the main libraries used in the code snippet you provided. Make sure you have these libraries installed if you plan to run the code. You can typically install them using package managers like pip or conda.

3.3.3. IDE Google Collab

Google Collab, short for Google Collaboratory, is a cloud-based development environment provided by Google. It allows users to write and execute Python code in a Jupiter Notebook-like interface directly on the cloud, without requiring any local installation. Collab provides free access to computational resources, including CPU, GPU, and even TPU (Tensor Processing Unit), enabling users to leverage the power of Google's infrastructure for their computational tasks[10].

3.3.4. Model implementation

This is the main code representation of our model that we explained before:

```
class GCN(nn.Module):
  def __init__(self, num_items, hidden_dim):
      super(GCN, self).__init__()
      self.embeddings = nn.Embedding(num_items, hidden_dim)
      self.conv1 = nn.Conv1d(hidden_dim, hidden_dim, kernel_size=2, padding=1)
      self.conv2 = nn.Conv1d(hidden_dim, hidden_dim, kernel_size=2, padding=1)
  def forward(self, adj, x):
      x = self.embeddings(x)
      # Permutes the dimensions of x to prepare it for the convolutional layers
      x = x.permute(1, 2, 0)
```

[6] https://pytorch.org/
[7] https://pandas.pydata.org/docs/getting_started/install.html
[8] https://scikit-learn.org/stable/install.html
[9] https://numpy.org/install/
[10] https://colab.research.google.com/?utm_source=scs-index

```
        # message passing
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        # Permutes the dimensions of x back to the original
        x = x.permute(2, 0, 1)
        # Aggregate messages from neighbors
        x = torch.matmul(adj, x)
        x = x.permute(1, 0, 2)
        x = x.mean(dim=1)
        return x
class SessionRec(nn.Module):
  def __init__(self, num_items, hidden_dim):
        super(SessionRec, self).__init__()
        self.gcn = GCN(num_items, hidden_dim)
        self.fc = nn.Linear(hidden_dim, num_items)
  def forward(self, adj, x):
        x = self.gcn(adj, x)
        x = self.fc(x)
        x = F.log_softmax(x, dim=1)
        return x
```

Listing: implementation of GCN-SBRS Model

3.4. Experiments and Analysis

In this section, we first describe the datasets and compared methods, and evaluation metrics used in the experiments. Then, we compare our proposed model GCN-SBRS with other comparative methods. Finally, we make a detailed analysis of GCN-SBRS under different experimental settings.

3.4.1. Datasets

We evaluate the effectiveness of our model on three standard transaction datasets: MOVILENS, YOOCHOOSE and DIGINETICA, which are publicly accessible and vary in terms of domain, size, and sparsity.

MOVILENS 1M:

MovieLens 1 million (ML-1M)[11] is one variant of the MovieLens dataset. It consists of approximately 1 million ratings given by 6,040 users to 3,706 movies. The ratings range from 1 to 5, with 5 being the highest rating. In addition to ratings, ML-1M also provides demographic information about the users, including age and occupation. The dataset is divided into three main files: ratings.dat (user-item ratings), users.dat (user information), and movies.dat (movie information).

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

MoviLens 100k:

MovieLens 100K (ML-100K)[12] is another popular variant of the MovieLens dataset. It is a smaller version compared to ML-1M and contains 100,000 ratings from 943 users on 1,682 movies. Like ML-1M, the ratings in ML-100K range from 1 to 5. The dataset is also divided into three main files: u.data (user-item ratings), u.user (user information), and u.item (movie information). ML-100K has been widely used as a benchmark for evaluating recommender systems due to its manageable size.

The MovieLens datasets can be obtained from the official MovieLens website or other platforms such as Kaggle and GitHub.

Yoochoose1/64:

The Yoochoose[13] dataset is a widely used e-commerce dataset that contains clickstream data from an online retail platform. It typically consists of user sessions, each containing a sequence of interactions (clicks) made by a user during their visit to the online store.

"Yoochoose1/64" refers to a specific fraction or subset of the original Yoochoose dataset; it might mean that it contains only a portion (1/64th) of the full dataset, possibly for the purpose of reducing the data size for experimentation or testing.

3.4.2. Data processing

- **Sorting the dataset**

The dataset is sorted based on two columns: 'user_id' and 'timestamp'.

Sorting the dataset in this order ensures that the data for each user is grouped together and ordered by timestamp.

- **Handling missing values**

The 'prev_item_id' column is created by shifting the 'item_id' column for each user by one position.

Rows with missing values (NaN) are dropped from the dataset using the dropna() function.

- **Limiting the dataset size**

The code limits the number of rows in the dataset to 15,000 by selecting the first 15,000 rows.

This step is useful for reducing the size of the dataset, which can be beneficial for faster execution or testing purposes.

- **Splitting the data into training and testing sets**

The train_test_split () function from scikit-learn is used to split the data into training and testing sets.

[12] https://grouplens.org/datasets/movielens/100k/
[13] https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015

The test_size=0.2 argument specifies that 20% of the data should be allocated for testing.

The random_state=42 argument fixes the random seed for reproducibility.

- **Mapping item IDs to indices**

Unique item IDs from both the training and testing data are obtained.

The np.union1d () function is used to find the union of the unique item IDs.

Item IDs are mapped to indices using a dictionary comprehension.

The 'item_idx' column is created in both the training and testing data by mapping the 'item_id' column to the corresponding item index.

- **Conversion to PyTorch tensors**

The data is converted to PyTorch tensors using the torch.LongTensor () function.

The 'item_idx' values are used as inputs, and the last 'item_idx' value is used as the target for each sequence.

3.4.3. Session Construction

We apply the same conditions to construct the sessions (min_sessions_lenght =3, max_sessions_lenght= 10 with time_threshold = 1hour).We summarizes the statistics of these datasets in Table1:

Table 3 - Datasets basic information

Dataset	Rows	Train Sessions	Test Sessions	Avr-L
MovieLens 100k	99057	6530	1970	4.50
MovieLens 1M	994169	99360	26530	4.29
Yoochoose1/64	371160	15919	2785	4.62

3.4.4. Training & Testing

- **Hyper parameters**

At this part we used the hyper parameters that were explored and optimized after using the grid search algorithm include the hidden dimension size (hidden_dim), learning rate (lr), and the number of epochs (num_epochs). These hyper parameters have a significant impact on the model's ability to capture complex patterns in the session data and make accurate predictions.

- **Loss function**

The NLLLoss (Negative Log Likelihood Loss) is a commonly used loss function in classification tasks. It measures the negative log likelihood of the predicted

probabilities for the correct class. It assumes that the model's output follows a softmax distribution, which converts the raw scores into probabilities that sum up to 1. The loss is calculated as the negative logarithm of the predicted probability assigned to the true class label.

Mathematically, for a single sample, the NLLLoss is calculated as follows:

$$\text{NLLLoss} = -\log(P(\text{correct\_class}))$$

Where:

P (correct_class) is the predicted probability assigned to the correct class by the softmax activation function.

The NLLLoss is commonly used as the loss function in the final layer of a neural network model for classification tasks. During training, the model aims to minimize this loss by adjusting its parameters to improve the probability assigned to the correct class and reduce the likelihood of misclassification.

- **Optimizer**

The Adam optimizer is an optimization algorithm commonly used for training neural networks. It is an extension of the stochastic gradient descent (SGD) algorithm that combines the benefits of adaptive learning rates and momentum.

The key idea behind the Adam optimizer is to adaptively adjust the learning rate for each parameter based on the estimated first and second moments of the gradients. This adaptive learning rate helps the optimizer converge faster and handle different types of parameters with varying magnitudes. Additionally, Adam incorporates the concept of momentum, which helps accelerate the learning process by accumulating the previous gradients.

- **Results**

This is the result we obtained after applying the same Hyper parameters and loss function and optimizer on the four datasets:

Table 4 – the accuracy results

Dataset	Train Accuracy/Loss	Test Accuracy/Loss	time
MovieLens 100k	0.7643/1.4574	0.7197/2.0935	2h
MovieLens 1M	0.6766/2.5720	0.8292 /1.4095	12h
Yoochoose1/64	0.6001/1.4984	0.1116/4.2414	3h

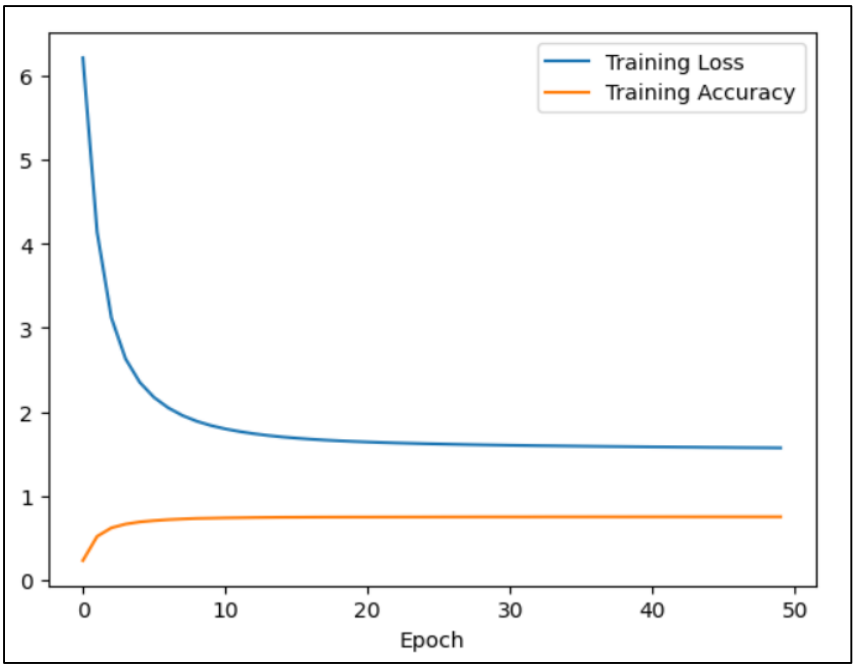


Figure 23 - MovieLens 100k train accuracy and train loss

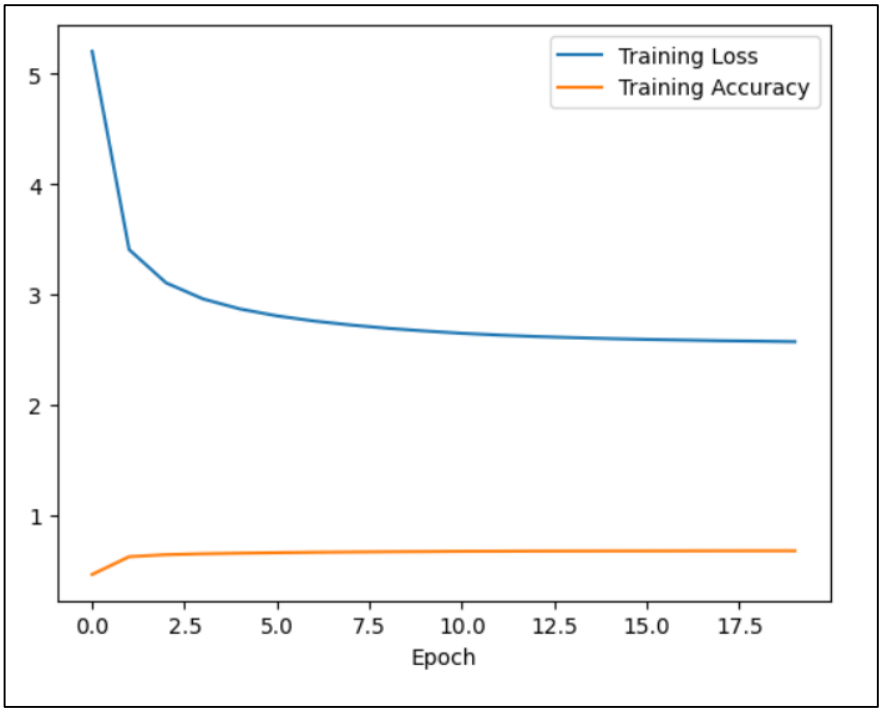


Figure 24 - MovieLens 1 million train accuracy and train loss

3.5. Evaluation Metrics

In our experiments, we use the following two widely-used evaluation protocols: Recall@K and MRR@K. By default, we set K=20.

- **Recall@20:** Recall@K represents the proportion of test cases which has the correctly recommended items in a top K position in a ranking list. In this paper, Recall@20 is used for all the tests, defined as:

$$Recall@20 = \frac{n_{hit}}{N},$$

Where N denotes the number of test data in the Session-based Recommender systems, n_{hit} denotes the number of cases which have the desired items in top K ranking lists, a hit occurs when t appears in the top K position of the ranking list of |I|

- **MRR@20:** The correct ranking of search results values in search results to evaluate the performance of the search system. The reciprocal rank is set to zero if the rank is above 20.

$$MRR@20 = \frac{1}{|I|}\sum_{t=1}^{|I|}\frac{1}{rank(i_t)}.$$

Where rank (it) is for the t -th item. The MRR is a normalized score of range [0, 1], an increase in its value reflects that the majority will appear higher in the ranking order of the recommendation list, which indicates a better performance of the corresponding recommender system.

3.5.1. Baseline

- POP: This baseline model recommends the top-N ranked items based on their popularity in the training data. It serves as a straightforward and robust baseline, especially in specific domains.
- S-POP: A variation of the baseline model that recommends the top-N most frequent items in both the entire training set and the current session.
- Item-KNN: This traditional item-to-item model suggests items similar to the ones already present in the user's history by calculating cosine similarity between candidate item A and existing item B.
- GRU4Rec [reference of the model]: Utilizing recurrent neural networks, GRU4Rec is designed for session-based recommendations. It adopts a session-parallel mini-batch training process and employs ranking-based loss functions during training.
- SR-GNN [reference of the model]: In this model, separate session sequences are aggregated into a graph structure, and Graph Neural Networks (GNNs) are applied to generate latent item vectors. Each session is then represented using a traditional attention network.

- GACOforRec [29]: Built upon GCNs, this algorithm accounts for user preferences in the application scenario. By incorporating Convolutional LSTM (ConvLSTM) and Orthogonal LSTM (ON-LSTM), it handles long-term and stable user preferences while preserving preference hierarchy.
- AUTOMATE: Keyed on the ARMAConv layer, AUTOMATE combines long-term preferences with current session interests to obtain graph transfer signals, resulting in personalized recommendations.

In this table we compare of different obfuscations in terms of their transformation capabilities:

Table 5 Comparison of metrics between different architectures

Model Class	Methods	MovieLens 1M		Yooshoose1/64		MovieLens 100k	
		Recall@20	MRR	Recall@20	MRR@20	Recall@20	MRR@20
Standard Baseline	POP	0.0646	0.0133	0.0671	0.001	0.1034	0.0209
	S-POP	0.0634	0.0132	0.3044	0.002	0.0776	0.0166
Traditional	Item-KNN	0.0016	0.0014	0.5660	0.003	0.0045	0.0033
Neural Networks	GRU4Rec	/	0.3041	0.6064	0.2289	/	/
	SR-GNN	/	0.3683	0.7003	0.3008	/	/
	GACOforRec	/	/	0.6879	0.2938	/	/
	AUTOMATE	/	/	0.7015	0.3072	/	/
Our Model	GCN-SBRS	**0.34**	**0.48**	**0.5154**	**0.1396**	**0.44**	**0.19**

As it shown in this table that our model performs very well with both MovieLens Datasets but with yoochoose gave as bad results and that because the differences between the two datasets MovieLens and Yoochoose:

Data Distribution: The MovieLens and Yoochoose datasets likely have different data distributions. These differences could affect how well our model generalizes from one dataset to another. Models trained on one dataset may not perform as well on a different dataset if the underlying data patterns are dissimilar.

Feature Engineering: The features (attributes) in the two datasets may have distinct characteristics. Our model might be well-suited to capturing the patterns present in the features of the MovieLens dataset but struggle to do so with the features in the Yoochoose dataset.

We aim to ameliorate these results in near future.

- **Result of the dataset MovieLens 100K**

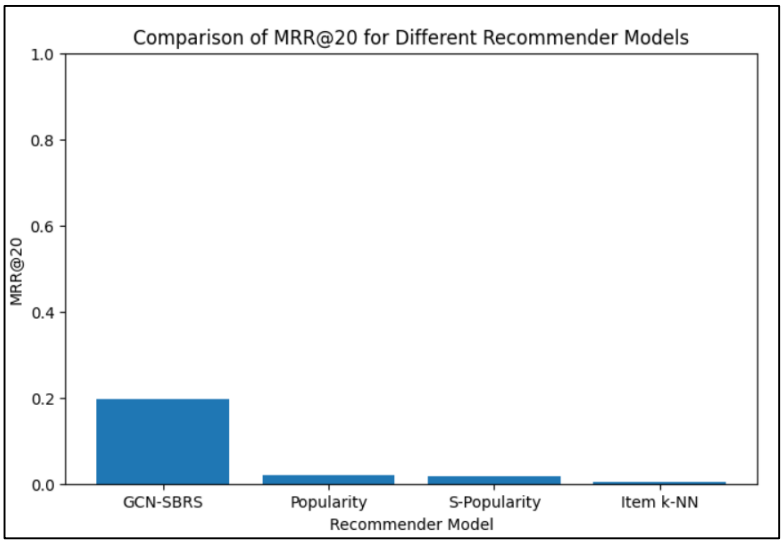


Figure 25 - Comparison of MRR@20 for different Recommender Models

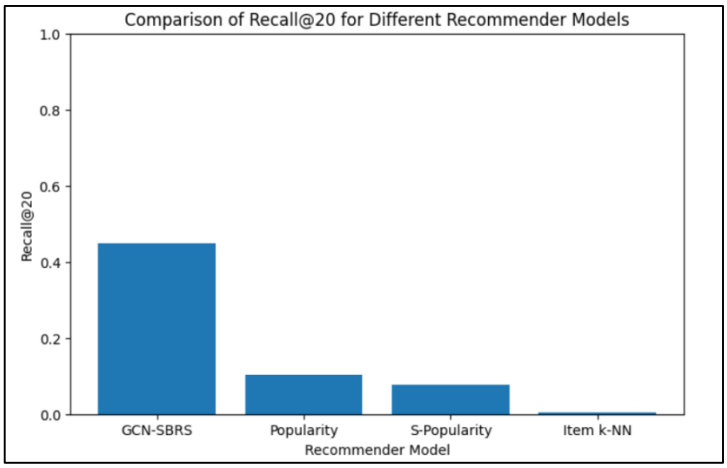


Figure 26 - Comparison of Recall@20 for Different Recommender Models

- **Result of the dataset MovieLens 1 million**

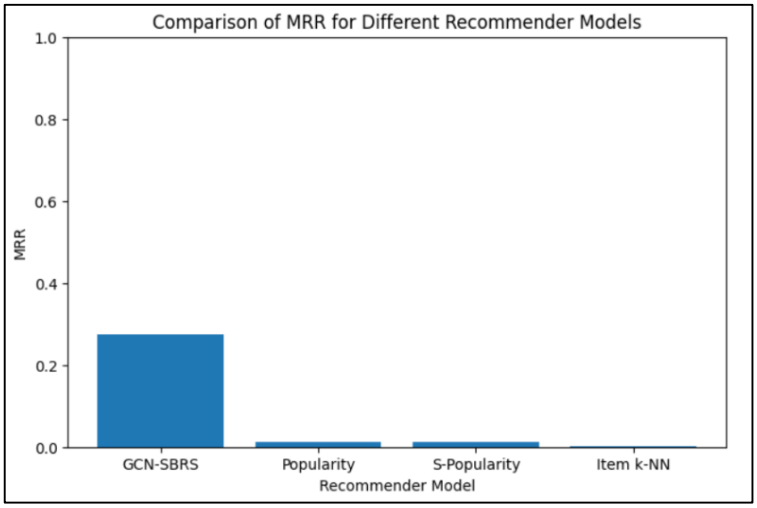


Figure 27 - Comparison of MRR for different Recommender model

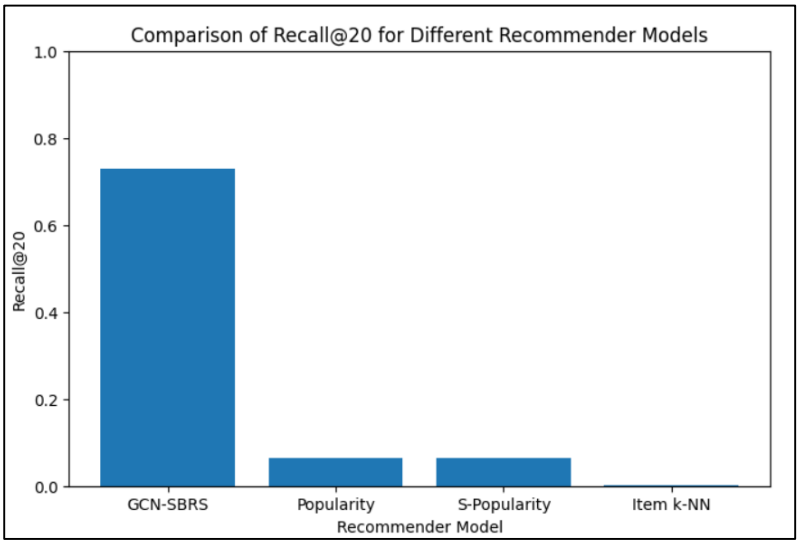


Figure 28 - Comparison of Recall@20 for different recommender models

3.6. Conclusion

In this chapter, we have discussed the underlying principle of our algorithm named GCN-SBRS, and provided an overview of its functionality. We then proceeded to present various experimental scenarios that were carefully designed to evaluate the performance of our algorithm. Finally, we thoroughly discussed the results obtained from these experiments, and we are pleased to report that they were highly satisfactory.

General Conclusion

A. Summary

In this thesis, the focus was on exploring the application of Graph Convolutional Networks (GCN) in Session-based Recommender Systems (SBRS). The research was conducted in three chapters. In Chapter 1, we introduced the concept of session-based recommender systems. Chapter 2 provided an in-depth understanding of Graph Convolutional Networks, showcasing their ability to capture graph-structured data. Lastly, in Chapter 3, we presented the design and implementation of an SBRS utilizing GCN, demonstrating its effectiveness in generating accurate and personalized recommendations based on users' session history.

The findings of this research highlight the potential of GCN in improving the performance of session-based recommender systems. By leveraging GCN, the SBRS was able to provide more accurate and personalized recommendations to users. The thesis contributes to the understanding of session-based recommender systems and demonstrates the applicability and effectiveness of GCN in this context.

B. Direction for future research

While this study has made a contribution to the field of session-based recommender systems using GCN, there are several avenues for future research:

Dataset Understanding: Thoroughly understand the differences between MovieLens and Yoochoose datasets with various appropriate metrics. Analyse their characteristics, feature distributions, and temporal aspects to identify potential sources of performance discrepancy. After that we will apply it on other datasets such as Diginetica…

Graph Convolutional Network Variants: Explore and compare various GCN variants (GraphSAGE, HyperGCN, …) or advanced graph neural network architectures to examine their suitability and effectiveness in session-based recommender systems. This includes investigating methods for handling the sparsity and scalability challenges often encountered in large-scale recommendation scenarios.

Hybrid Recommender Systems: Investigate the integration of GCN-based session-based recommender systems with other recommendation approaches, such as collaborative filtering or content-based methods. This hybrid approach may leverage the strengths of different algorithms to provide more accurate and diverse recommendations to users.

Real-world Deployment and Evaluation: Conduct extensive evaluations of the developed SBRS using GCN in real-world scenarios, considering factors such as scalability, robustness, and user satisfaction. Additionally, perform comparative studies with other state-of-the-art session-based recommendation algorithms to establish benchmarks and further validate the effectiveness of GCN-based approaches.

By exploring these research directions, future studies can contribute to the advancement and practical implementation of session-based recommender systems using GCN, ultimately enhancing the user experience and providing valuable recommendations in various domains and applications.

Bibliography

[1] G. T. A. Adomavicius, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering,* vol. 17, no. 6, pp. 734-749, 2005.

[2] T. S. ·. M. Z. ·. X. L. ·. Y. L. ·. S. Ma, "How good your recommender system is? A survey on evaluations," *International Journal of Machine Learning and Cybernetics,* 2017.

[3] B. Schwartz, The paradox of choice: why less is more, New York: Ecco, 2004.

[4] F. R. ·. L. R. ·. B. S. ·, Recommender Systems, 2011.

[5] L. C. Y. W. Q. Z. S. M. A. O. L. SHOUJIN WANG, *A Survey on Session-based Recommender Systems,* 2021.

[6] L. Cao, "Coupling learning of complex interactions," *Information Processing & Management,* vol. 51, no. 2, p. 167–186, 2015.

[7] L. Cao, "Non-IID recommender systems: a review and framework of recommendation paradigm shifting," *Engineering,* vol. 2, no. 2, p. 212–224, 2016.

[8] "Recommender Systems: Behind the Scenes of Machine Learning-Based Personalization," altexsoft, 27 july 2021. [Online].

[9] G. T. A. Adomavicius, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering,* vol. 17, pp. 734-749, 2005.

[10] Z. A. a. I. U. Shah Khusro, "Recommender Systems: Issues, Challenges,".

[11] M. Madhukar, "Challenges & Limitation in Recommender," *IBM India Pvt. Ltd.,* p. 5.

[12] Turing, "A Guide to Content-Based Filtering In Recommender Systems," [Online]. Available: https://www.turing.com/kb/content-based-filtering-in-recommender-systems.

[13] A. Kumar, "Recommender Systems in Machine Learning: Examples," 26 May 2023. [Online]. Available: https://vitalflux.com/recommender-systems-in-machine-learning-examples/.

[14] M. L. B. D. L. R. J. Montaner, "A taxonomy of recommender agents on the internet," *Artificial Intelligence Review,* vol. 19, pp. 285-330, 2003.

[15] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction,* vol. 12, pp. 331-370, 2002.

[16] J. Wu, "Knowledge-Based Recommender Systems: An Overview," 27 mai 2019. [Online]. Available: https://medium.com/@jwu2/knowledge-based-recommender-systems-an-overview-536b63721dba.

[17] D. .. R. M.Sridevi, "A Simple and Efficient Demographic," vol. 10, no. 7, 2017.

[18] G. M. B. R. F. T. A. Adomavicius, "Context-Aware Recommender Systems," *AI Magazine,* vol. 32, no. 3, 2011.

[19] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing,* vol. 1, no. 5, 2001.

[20] M. T. W. L. Gunnar Schröder, "Setting Goals and Choosing Metrics for Recommender," p. 8.

[21] Z. Deutschman, "Recommender Systems: Machine Learning Metrics and Business Metrics," neptune.ai, 19 april 2023. [Online]. Available: https://neptune.ai/blog/recommender-systems-metrics.

[22] Z. Ahmed, "Automatic Evaluation of Recommendation Systems: Coverage, Novelty and Diversity," MLearning.ai, 26 february 2021. [Online]. Available: https://medium.com/mlearning-ai/automatic-evaluation-of-recommendation-systems-coverage-novelty-and-diversity-cc140330d3e7#:~:text=Diversity%20is%20a%20measure%20of,directed%20by%20the%20same%20director..

[23] L. G. M. L. P. S. G. F. M. M. P. Iaquinta, "Introducing serendipity in a content-based recommender system," in *Proceedings of the 2008 8th international conference on hybrid intelligent systems (HIS '08)*, 2008.

[24] F. R. ·. L. R. ·. B. S. ·, Recommender Systems.

[25] V. Ankam, "Limitations of a recommendation system," O'Reilly learning platform, [Online]. Available: https://www.oreilly.com/library/view/big-data-analytics/9781785884696/ch08s02.html.

[26] C. C. Y.-S. Shahabi, Web information personalization: challenges and approaches, Springer, 2003, pp. 5-15.

[27] X. K. T. Su, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence,* no. 4, 2009.

[28] I. K. C. B. A. P. H. Gunes, "Shilling attacks against recommender systems: a comprehensive survey," *Artificial Intelligence Review,* no. 42, pp. 767-799, 2014.

[29] Unknown, "7 Critical Challenges of Recommendation Engines," appier, 2 2 2021. [Online]. Available: https://www.appier.com/en/blog/7-critical-challenges-of-recommendation-engines.

[30] Z. H. T. C. Y. RUIHONG QIU, *Exploiting Positional Information for Session-based,*

2021.

[31] L. C. S. W. e. a. Liang Hu, "Diversifying personalized recommendation with user-session context," in *IJCAI*, 2017.

[32] L. H. L. C. Shoujin Wang, "Perceiving the next choice with comprehensive transaction embeddings for online recommendation," in *ECML-PKDD*, 2017.

[33] A. K. L. B. T. Balazs Hidasi, *SESSION-BASED RECOMMENDATIONS WITH RECURRENT NEURAL NETWORKS,* 2016.

[34] E. S. A. C. Flavian Vasile, "Meta-Prod2Vec: product embeddings using side-information for recommendation," in *RecSys*, 2016.

[35] Y. T. Y. Z. e. a. Shu Wu, "Session-based recommendation with graph neural networks," in *AAAI*, 2019.

[36] A. K. Balázs Hidasi, "Recurrent neural networks with top-k gains for session-based recommendations," in *CIKM*, 2018.

[37] e. a. Massimo Quadrana, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in *RecSys*, 2017.

[38] L. C. L. H. S. B. X. H. L. X. W. L. "Shoujin Wang, "Jointly modeling intra- and inter-transaction dependencies with hierarchical attentive transaction embeddings for next-item recommendation," *IEEE Intelligent Systems,* pp. 1-7, 2020.

[39] L. C. S. W. e. a. Liang Hu, "Diversifying personalized recommendation with user-session context.," in *IJCAI*, 2017.

[40] L. H. L. C. e. a. Shoujin Wang, "Attention-based transactional context embedding for next-item recommendation.," in *AAAI*, 2018.

[41] L. H. Y. W. e. a. Shoujin Wang, "Sequential recommender systems: challenges, progress and prospects," in *IJCAI*, 2019.

[42] A. K. e. a. Fajie Yuan, "A simple convolutional generative network for next item recommendation.," in *WSDM*, 2019.

[43] K. W. Jiaxi Tang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *WSDM*, 2018.

[44] A. K. e. a. Fajie Yuan, "A simple convolutional generative network for next item recommendation," in *WSDM*, 2019.

[45] H. Z. Q. L. Z. H. T. M. E. C. Zhi Li, "Learning from history and present: next-item recommendation via discriminatively exploiting user behaviors," in *SIGKDD*, 2018.

[46] D. Y. Y. X. Wenjing Meng, "Incorporating user micro-behaviors and item knowledge into multi-task learning for session-based recommendation," in *SIGIR*, 2020.

[47] K. Suzuki, ARTIFICIAL NEURAL NETWORKS † METHODOLOGICAL ADVANCES AND BIOMEDICAL APPLICATIONS, India, March, 2011.

[48] S. Kumar, "Supervised vs Unsupervised vs Reinforcement," 29 january 2020. [Online]. Available: https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/.

[49] R. S. a. B. A. G. Sutton, Reinforcement Learning: An Introduction, MIT Press, 2018.

[50] L. P. L. M. L. a. M. A. W. Kaelbling, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research,* vol. 4, pp. 237-285, 1996.

[51] harkiran78, "Artificial Neural Networks and its Applications," [Online]. Available: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/.

[52] G. Singh, "Introduction to Artificial Neural Networks," 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/09/introduction-to-artificial-neural-networks/.

[53] Y. M. a. J. Tang, Deep Learning on Graphs.

[54] Y. H. P. &. S.-S. S. P. Jinming Zou PhD, "Overview of Artificial Neural Networks," [Online].

[55] P. Bahati, "Activation Functions in Neural Networks [12 Types & Use Cases]," 27 March 2021. [Online]. Available: v7labs.com/blog/neural-networks-activation-functions#:~:text=drive%20V7's%20tools.-,What%20is%20a%20Neural%20Network%20Activation%20Function%3F,prediction%20using%20simpler%20mathematical%20operations..

[56] J. Z. Zhiyuan Liu, Introduction to Graph Neural Networks, the Morgan & Claypool Publishers series, 2020.

[57] S. S. S. S. Anidhya Athaiya, "ACTIVATION FUNCTIONS IN NEURAL," *IJEAST,* vol. 4, no. 12, pp. 310-316, April 2020.

[58] S. SHARMA, "Activation Functions in Neural Networks," 6 Sep 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[59] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[60] I. B. Y. a. C. A. Goodfellow, Deep Learning, MIT Press, 2016.

[61] I. Mayachita, "Understanding Graph Convolutional Networks for Node Classification," Towards Data Science, 10 june 2020. [Online]. Available: https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b.

[62] A. Graves, "Generating Sequences with Recurrent Neural Networks," 2013. [Online]. Available: arXiv:1308.0850.

[63] S. a. S. J. Hochreiter, " Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[64] K. V. M. B. B. D. a. B. Y. Cho, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," arXiv preprint, 2014. [Online]. Available: arXiv:1409.1259.

[65] A. Biswal, "Recurrent Neural Network(RNN) Tutorial: Types, Examples, LSTM and More," 10 April 2023. [Online]. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn.

[66] M. Nielsen, Neural Networks and Deep Learning, Determination Press.

[67] R. M. T. a. B. Y. Pascanu, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2013.

[68] [Online]. Available: Upgrad.com.

[69] GeeksforGeeks, "Introduction to Convolution Neural Network," [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/.

[70] Y. B. Y. a. H. G. LeCun, "Deep learning," *Nature,* vol. 521, no. 7553, pp. 436-444, 2015.

[71] A. S. I. a. H. G. E. Krizhevsky, "ImageNet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012.

[72] V. a. H. G. E. Nair, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[73] Y. B. L. B. Y. a. H. P. LeCun, " Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[74] D. M. A. a. B. S. Scherer, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, 2010.

[75] K. a. Z. A. Simonyan, "Very deep convolutional networks for large-scale image recognition," arXiv preprint , [Online]. Available: arXiv:1409.1556.

[76] N. &. Z. H. &. P. M. &. P. J. &. S. M. Herakovic, "Distributed Manufacturing Systems with Digital Agent," no. 65, pp. 650-657, 2019.

[77] N. Biggs, Algebraic Graph Theory, Cambridge University Press, 1993.

[78] UC Berkeley data structures course, "Graphs," [Online]. Available:

https://guides.codepath.com/compsci/Graphs#introduction.

[79] F. Chung, L. Lu and V. Vu, "Spectra of random graphs with given expected degrees," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 100, no. 11, p. 6313–6318, 2003.

[80] W. L. Hamilton, R. Ying and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin,* vol. 40, no. 3, pp. 52-74, 2017.

[81] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks,* vol. 20, no. 1, pp. 61-80, 2009.

[82] C. Cangea, M. Aziz, M. Hinz and T. Lukasiewicz, " Neural networks for graphs: A survey," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 30, no. 11, pp. 3243-3258, 2018.

[83] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu and M. Sun, "Graph neural networks: A review of methods and applications," arXiv preprint, [Online]. Available: https://arxiv.org/abs/1812.08434.

[84] M. Zhang, Z. Cui, M. Neumann and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[85] M. Zitnik, M. Agrawal and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks.," *Bioinformatics,* vol. 34, no. 13, pp. i457-i466., 2018.

[86] A. Mashaghi and e. al., "Investigation of a protein complex network," *European Physical Journal B,* vol. 41, no. 1, pp. 113-121, 2004.

[87] P. Shah, A. Ashourvan, F. Mikhail, A. Pines, L. Kini, K. Oechsel, S. R. Das, J. M. Stein and R. T. Shinohara, " Characterizing the role of the structural connectome in seizure dynamics," *Brain,* vol. 142, no. 7, p. 1955–1972, 2019-07-01.

[88] T. Adali and A. Ortega, "Applications of Graph Theory [Scanning the Issue]," *Proceedings of the IEEE,* vol. 106, no. 5, p. 784–786, May 2018.

[89] A. Mishra, "Application of Graph Data Structure," [Online].

[90] Wikipedia, "Graph theory," [Online]. Available: https://en.wikipedia.org/wiki/Graph_theory#.

[91] K. H. Rosen, Discrete mathematics and its applications, New York: McGraw-Hill, 2011-06-14.

[92] P. Sharma, "What are Graph Neural Networks, and how do they work?," 1 March 2022 . [Online]. Available: https://www.analyticsvidhya.com/blog/2022/03/what-

are-graph-neural-networks-and-how-do-they-work/.

[93] A. A. Awan, "A Comprehensive Introduction to Graph Neural Networks (GNNs)," Jul 2022. [Online]. Available: https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial.

[94] T. N. a. W. M. Kipf, "Semi-Supervised Classification with Graph Convolutional Networks.," in *Proceedings of the International Conference on Learning*, 2017.

[95] Z. P. S. C. F. L. G. Z. C. a. Y. P. S. Wu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 32, no. 1, pp. 4-24, 2020.

[96] D. K. Hammond, P. Vandergheynst and R. Gribonval, "Wavelets on Graphs via Spectral Graph Theory," *Applied and Computational Harmonic Analysis,* pp. 129-150, 2011.

[97] A. ÖZCAN1*, "Applying graph convolution networks to recommender systems based on graph," 12 April 2022.

[98] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 32, no. 1, pp. 4-24, 2020.

[99] W. Y. R. a. L. J. Hamilton, "Inductive Representation Learning on Large Graphs," in *Advances in Neural*, 2017.

[100] C. Pham, "Graph Convolutional Networks (GCN)," 22 10 2020. [Online]. Available: https://www.topbots.com/graph-convolutional-networks/?fbclid=IwAR2fg-900x7AsvDyQbW1XADToJGnTZqsxVwFJV_TZ7O-Thadavpt5lVjZd0.

[101] M. Zhang, Y. Chen and M. Li, "Large-scale and Deep Graph Convolutional Networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[102] P. C. G. C. A. R. A. L. P. a. B. Y. Velickovic, "Graph Attention Networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[103] M. B. X. a. V. P. Defferrard, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[104] K. H. W. L. J. a. J. S. Xu, "How Powerful are Graph Neural Networks?," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[105] Y. Liu, Y. Liu and C. Yang, "Modulation recognition with graph convolutional network," *IEEE Wireless Communications Letters,* vol. 9, no. 5, pp. 624-627, 2020.

[106] Z. W. S. A. L. Y. Bruna J, "Spectral networks and locally connected networks on graphs," 2013.

[107] L. S. G. R. L. a. P. V. Bruno, "A Comprehensive Review of Graph Convolutional Networks," *Applied Sciences,* vol. 10, no. 21, p. 7685, 2020.

[108] A. K. L. B. D. T. Balázs Hidasi, "Session-based recommendations with recurrent neural networks," in *ICLR*, 2016.

[109] E. S. A. C. Flavian Vasile, "Meta-Prod2Vec: product embeddings using side-information for recommendation," in *RecSys*, 2016.

[110] H. T. J. X. a. R. M. Si Zhang1*, "Graph convolutional networks:," 2019.

[111] B. D. M. J. R. E. S. J. B. M. Monti F, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *CVPR*, 2017.

[112] B. X. V. P. Defferrard M, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016.

[113] B. J. Garcia V, "Few-shot learning with graph neural networks," 2017.

[114] C. Y. L. X. W. H. Z. Y. X. E. Kampffmeyer M, "Rethinking knowledge graph propagation for zero-shot learning," 2018.

[115] L. S. S. A. Narasimhan M, "Out of the box: reasoning with graph convolution nets for factual visual question answering," in *Advances in neural information processing systems*, 2018.

[116] X. C. Z. W. Y. J. Cui Z, "Context-dependent diffusion network for visual relationship detection," in *2018 ACM multimedia conference on multimedia conference*, New York, 2018.

[117] P. Y. L. Y. M. T. Yao T, "Exploring visual relationship for image captioning," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[118] Z. Y. C. C. F.-F. L. Xu D, "Scene graph generation by iterative message passing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[119] Z. Y. L. D. Dai B, "Detecting visual relationships with deep relational networks," in *Proceedings of the IEEE conference on computer vision and Pattern recognition*, 2017.

[120] X. Y. L. D. Yan S, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[121] G. A. Wang X, " Videos as space-time region graphs," in *Proceedings of the European conference on computer vision (ECCV)*.