PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

IBN KHALDOUN UNIVERSITY OF TIARET

THESIS

Presented to:
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

To obtain the:
MASTER'S DEGREE

Major: Software Engineering

Presented by:
AIT ARAB SARA MELISSA
GAFOUR MOHAMED ALI

On the theme:

Gated Graph Neural Networks for Building Session-Based Recommender Systems

Publicly presented on 10/07/2023 in Tiaret to the jury consisting of:

Mr. Alem Abdelkader	MAA	University of Tiaret	Chairman
Mr. Boudaa Boudjemaa	MCA	University of Tiaret	Supervisor
Mrs. Boubekeur Aicha	MAA	University of Tiaret	Examiner

2022-2023

Acknowledgements

We would like to express our profound gratitude to God, the Most Gracious and the Most Merciful, for giving us the opportunity and the ability to complete this thesis. We are deeply indebted to our families, especially our parents, for their unconditional love, encouragement, and support throughout our academic journey. We are also immensely thankful to each other, for our valuable collaboration, insights, and feedback that greatly improved the quality of this work. Furthermore, we would like to acknowledge the guidance and supervision of Professor Boudaa Boudjemaa, who provided us with constructive criticism, useful suggestions, and constant motivation during this research project. We would also like to thank the members of the jury for their time and attention to our work. Last but not least, we would like to extend our heartfelt appreciation to our friends and loved ones who have stood by us through thick and thin, offering words of encouragement, understanding, and a shoulder to lean on. We are grateful beyond words.

Ibn Khaldoun University – Tiaret

July, 2023

Mohamed Ali Gafour

Sara Melissa Aït Arab

Abstract

In the modern world, consumers are faced with an overwhelming number of options and alternatives in various domains and applications. This phenomenon, known as choice overload, can cause confusion, dissatisfaction, and decision paralysis among consumers. Therefore, there is a growing demand for recommendation systems that can assist consumers in filtering out irrelevant options and guiding them to the most suitable ones based on their preferences and needs. Recommendation systems are software tools and techniques that provide personalized suggestions for items that users may like or need. They have become essential in various domains and applications, namey, e-commerce, streaming media, news platforms, and social media platforms. They can enhance user satisfaction, loyalty, and retention, as well as increase sales, conversion, and revenue for businesses. Session-based recommendation is a challenging task that aims to predict user actions based on anonymous sessions. Existing methods use sequential models to learn user and item representations for recommendation. However, these methods have limitations in capturing accurate user vectors in sessions and modeling complex item transitions. In this thesis, we propose a novel method called L-GGNN-ATT, which stands for **L**ossless **G**ated **G**raph **N**eural **N**etworks with **Att**ention as Readout. L-GGNN-ATT treats session sequences as graph-structured data and applies graph neural network – which is a deep learning method to infer on graphs – to learn item embeddings that can reflect complex item transitions. Moreover, it uses an attention network to represent each session as a combination of the global preference, i.e, the various item interactions, and the current interest of that session, which is simply the current item we are on. We conduct extensive experiments on two real datasets and demonstrate that our model significantly outperforms the state-of-the-art methods for session-based recommendation.

Keywords – session-based recommendation, gated graph neural networks, graph representation learning, sequential data modeling, user behavior analysis

Contents

List of Figures

List of Tables

Acronyms

RS Recommender Systems
SBRS Session-Based Recommender Systems
DL Deep Learning
ML Machine Learning
RNN Recurrent Neural Networks
GNN Graph Neural Networks
GGNN Gated Graph Neural Networks
KNN K-Nearest Neighbors
LSTM Long Short-Term Memory
GRU Gated Recurrent Units
MLP Multi-layer Perceptron
CNN Convolutional Neural Networks
AI Artificial Intelligence
NN Neural Networks
NLP Natural Language Processing
MPNN Message-Passing Neural Networks
GAT Graph Attention Networks
GCN Graph Convolutional Networks
MRR Mean Reciprocal Rank

CHAPTER:
General Introduction

General Introduction

Background

Recommender Systems (RSs) are information filtering systems that provide *suggestions* for items that are most relevant to a particular user. They have been widely used in various domains like e-commerce, entertainment, social media, and education, to help users cope with the information overload and discover items of their interest [1].

The first concrete appearances of RSs can be traced back to the early 1990s, when systems like Tapestry [2] and GroupLens were developed to filter and recommend news articles based on user ratings. Since then, RSs have evolved and diversified to handle different types of data – such as ratings, reviews, clicks or interactions, and browsing history – to recommend different kinds of items like products, books, news, friends.

While the conventional methods rely on user identifiers to track the long-term preferences of individual users across multiple sessions, Session-Based Recommender Systems (SBRS) only use anonymous session identifiers to model the short-term preferences of anonymous users within a single session [2]. Therefore, SBRSs can only use short-term information to make general recommendations. However, SBRSs have the advantage of being able to handle new users and items without requiring any prior information or feedback.

Deep Learning (DL) – which is a branch of machine learning – has achieved remarkable success in many domains. For example, computer vision, natural language processing, and speech recognition. In recent years, DL has also been applied to RSs and has shown promising results in improving the performance and functionality of RSs. Session-based recommender systems can be divided into two main phases: the *model-free* phase from the late 1990s to the early 2010s, and the *model-based* phase from the early 2010s onwards. The former phase relied on data mining methods: pattern mining, association rule mining, and sequence mining, to discover frequent or sequential patterns and rules from user sessions and use them for recommendation. This phase reached its peak in the mid-2000s, when many pattern/rule-based and sequence-based approaches were proposed. The latter phase leveraged statistical and machine learning methods, especially time-series models – markov chain models and recurrent neural network (RNN) models to mention a few – to learn latent features and temporal dependencies from user sessions and use them for recommendation [3]. This phase has been flourishing since 2017, due to the rapid advancement of deep learning techniques. Many novel neural models have been developed for predicting the next item or basket in user sessions in recent years.

Graphs are powerful data structures that can capture the complex and rich relationships among entities in various domains. By representing data as graphs, we can preserve the structural and semantic information that might be lost or distorted in other forms of representation. Graph Neural Networks (GNNs) are a class of deep learning models that can operate on graph-structured data by propagating and aggregating information along the edges of a graph. GNNs have shown remarkable

performance and versatility in solving problems involving sequential data, such as natural language processing, computer vision, social network analysis, and pertinently, recommender systems. GNNs can exploit the sequential patterns and dependencies among nodes in a graph, as well as the node features and attributes, to learn expressive and informative node embeddings for various tasks [4].

The first work that applied GNNs to session-based recommendation was proposed by Wu et al. in 2019 [5]. They used a Gated Graph Neural Network (GGNN) to encode user sessions and an attention mechanism to compute the relevance scores between items. Since then, many variants and extensions of GNNs have been proposed for session-based recommendation. These works have shown that GNNs can outperform other DL models for session-based recommendation in terms of accuracy and diversity.

Problem Statement

The machine learning and recommender systems community has overlooked the problem of session-based recommendation. Many e-commerce recommender systems (mainly those of small retailers) and most of the news and media sites lack effective methods to track the users who visit their sites for a long duration. Cookies and fingerprinting can offer some help, but they are often unreliable and raise privacy concerns. Even when tracking is feasible, many users only have one or two sessions on a smaller e-commerce site, and in some domains the user behavior is more influenced by the session than by the user. Therefore, different sessions of the same user should be handled independently. Most session-based recommendation systems that are deployed for e-commerce use relatively simple methods that do not leverage a user profile. However, these methods are restricted because they only focus on the last action of the user and disregard the information of previous actions.

Graph neural networks (GNNs) are a suitable choice for session-based recommendation, as they can handle graph-structured data and learn meaningful node embeddings. GNNs can model user sessions as graphs, where nodes are items and edges are interactions or transitions between items. In short, the questions that motivation this thesis are:

How can we design effective and efficient GGNN models for session-based recommendation? And what are the challenges and opportunities in this emerging research area?

Delimitation

We focus on the problem of session-based recommendation, which aims to predict user actions based on anonymous sessions without user profiles. Most existing methods for session-based recommendation are based on sequential models that have limitations in capturing the complex transitions and dependencies among items in a session.

Naturally, this thesis delimits its scope to the following aspects:

- We exclusively consider session-based recommendation without user profiles, and only address other types of recommendation problems, such as collaborative filtering, content-based filtering, or hybrid filtering for origination purposes.
- The thesis only considers item recommendation as the target task, and does not address other tasks that can be related to session-based recommendation, such as *next-basket prediction*, *session segmentation*, or *session clustering*.
- Although other deep learning methods can be involved in the model (e.g. attention mechanisms or embedding layers), we still consider GNN as the core component of the model that determines its performance and functionality.
- The thesis only evaluates the model on two real-world datasets from e-commerce, and does not test it on other domains or synthetic datasets that may have different characteristics or challenges.

Approach

To tackle this task, a model that produces session-based recommendations based on graph neural networks is presented. The user sessions are modeled as graphs, where nodes are items and edges are interactions or transitions between items. The GNN model takes the session graph as input and learns expressive and informative node embeddings that capture both the item features and the sequential patterns. The GNN model then uses an attention network to represent each session as the composition of the global preference and the current interest of that session. The GNN model then outputs a recommendation that depends on the session representation. A specific variation of the SR-GNN [5] will be customized to fit this task. The challenges and opportunities of using GNNs for session-based recommendation will be discussed and analyzed.

Outline

This thesis consists of three chapters in addition to a general introduction and a general conclusion:

- **GENERAL INTRODUCTION**
  A brief introduction to recommender systems and the context, research questions, and scope of the thesis.
- **CHAPTER ONE: SESSION-BASED RECOMMENDER SYSTEMS**
  An introductory exploration into the theoretical aspects of recommender systems, delving into the domain of sequential recommender systems, with the primary objective of establishing a fundamental groundwork and offering a comprehensive perspective on session-based recommender systems.
- **CHAPTER TWO: GATED GRAPH NEURAL NETWORKS**
  This chapter provides essential contextual information and foundational knowledge pertaining

to artificial intelligence, machine learning, and deep learning. It aims to familiarize with the fundamental concepts required to effectively engage with deep learning and Gated Graph Neural Networks (GGNNs).

- **CHAPTER THREE: SESSION-BASED RECOMMENDER SYSTEMS USING GATED GRAPH NEURAL NETWORKS**
  We delve deeply into the practical implementation of gated graph neural networks within the development of a session-based recommendation system. Firstly, it introduces the evaluation metrics employed for the experiment, followed by the description of the utilized datasets and baselines. Finally, a comparative analysis is conducted between the results of the baselines and the proposed GGNN-based model, accompanied by a comprehensive discussion of the obtained outcomes.

- **GENERAL CONCLUSION**
  In light of the findings and the discourse presented in the preceding chapter, this final section pertains to the research inquiries and provides a concise overview of the present study. Furthermore, potential avenues for future research are expounded upon and explored.

CHAPTER ONE:

Session-Based Recommender Systems

1 Session-Based Recommender Systems

1.1 Introduction

Recommender Systems (RSs) are machine learning based systems that aim to provide personalized suggestions for items or services that are most relevant and useful for a given user. They have become ubiquitous in various domains such as e-commerce, entertainment, education, and social media, where they help users cope with the problem of information overload and discover new products or content.

Recommender Systems research has been evolving rapidly in the past decades, with many different approaches and techniques being proposed and applied. Some of the main research areas include traditional RSs: content-based, collaborative filtering, and hybrid systems [6], which rely on historical interactions and user/item attributes; non-traditional RSs: context-aware [7], sequential systems [8], and social network-based [9], which leverage other sources of information besides historical data; and evaluation methods and metrics for RSs, which assess their performance and quality from different perspectives [10].

One of the emerging research areas in RSs is Session-Based Recommender Systems (SBRSs). SBRSs differ from traditional recommender systems in that they do not require any prior knowledge or explicit feedback from the users. Instead, they generate recommendations based on the user's current session behavior (i.e. clicks or views). SBRSs are especially suitable for scenarios where users are anonymous or have dynamic preferences that change over time [3]. They can also capture short-term user intents and provide more timely and diverse recommendations.

In this chapter, we will explore different types of recommender systems, including traditional, non-traditional, and session-based approaches. We will discuss their underlying principles, advantages, and limitations, and highlight some of the recent advancements in the field. Additionally, we will provide an overview of evaluation methods and metrics for recommender systems and introduce Session-Based Recommender Systems.

1.2 Recommender Systems

A Recommender System (RS) is a type of information filtering system that provides suggestions for items that are most relevant to a particular user. The items can be anything: products, movies, music, books, or online content. The suggestions are based on various sources of information like user preferences, ratings, feedback, behavior, or item features. The output of a Recommender System is typically a ranked list of items that the user might like or benefit from [11].

> "A lot of times, people don't know what they want until you show it to them." — Steve Jobs

An RS is considered traditional (otherwise conventional) if it is based on historical interactions

and user/item attributes. Traditional RSs include *content-based* and *collaborative filtering* systems, which can be mixed together in what is called a hybrid Recommender System. They are passive in that they try to adapt their recommendations to the user's historical interests [12]. Conversely, if an RS uses other sources of information besides historical interactions and user/item attributes, it is considered non-traditional. Non-traditional recommender systems may use semantic approaches, social networks, context and time awareness, or other techniques to enhance their recommendations (see figure 1.1) [3]. They are also called advanced or modern recommender systems.

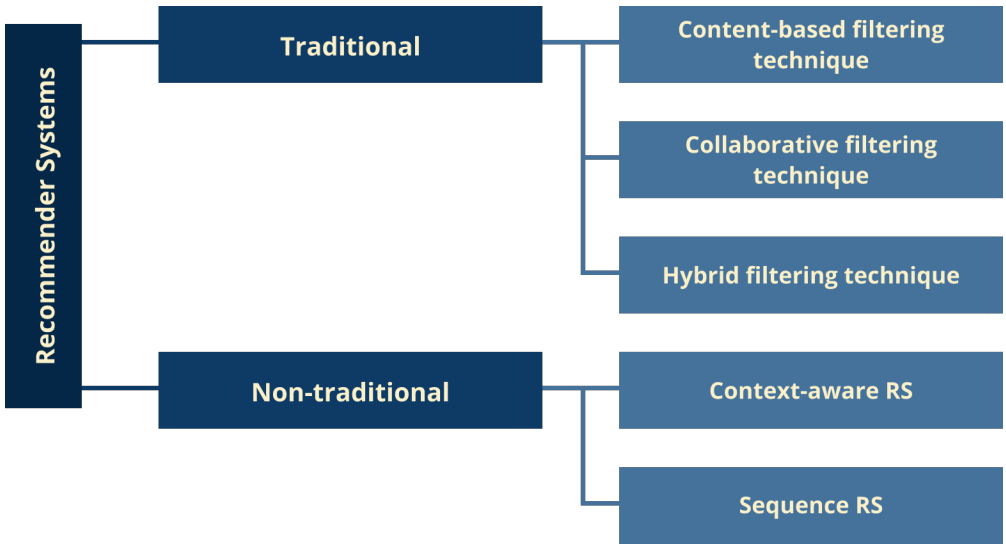

Figure 1.1: Nonexhaustive classification of Recommender Systems

1.2.1 Traditional Recommender Systems

Before exploring other types of RSs, let us first review the traditional ones. These are based on historical interactions and user/item attributes.

1.2.1.1 Content-based Filtering

A content-filtering RS suggests items that match the user's preferences based on the features or attributes of the items. For instance, a user who likes movies can get recommendations from a content-filtering recommender system that uses keywords (like genre, director, actors, etc.) to describe the movies and to find out what kind of movies the user enjoys. In essence, content-filtering RSs work well when there is a clear idea of what is desired and when the items have enough information to describe them. As illustrated in the leftmost side of figure 1.2, information about the user has very little influence [10, 13].

High-order Architecture

1. A content analyzer that extracts relevant features from each item. Features are extracted from various sources to be converted into a keyword-based vector-space representation. This

is the first step of any content-based recommendation system, and it is highly domain-specific [13].

2. A profile learner that builds and updates a user profile based on their preferences and feedback. A content-based model that is specific to a given user is constructed to predict user interests in items based on their past history of either buying or rating items. To achieve this goal, user feedback is leveraged, which may be manifested in the form of previously specified ratings (explicit feedback) or user activity (implicit feedback). Such feedbacks are used in conjunction with the attributes of the items to construct the training data. The resulting model is referred to as the *user profile* because it conceptually relates user interests (ratings) to item attributes [13].

3. A filtering component that generates recommendations by matching the user profile with the item features. This component can use similarity measures (cosine similarity, Jaccard index, etc.) to rank and select the most relevant items for each user [13].

Advantages

1. They are user independent (see subsection 1.2.3), meaning they can provide personalized recommendations even if there are few or no other users in the system. They also do not have a cold-start problem (see subsection 1.2.3), meaning they can recommend new items right away without waiting for user feedback [11].

2. They allow users to have more control over their recommendations by providing them with explanations or filters based on item features. They also respect users' privacy by not sharing their data with other users [11].

Drawbacks

1. They are limited by content analysis, meaning they can only recommend items that have sufficient and accurate features available. They may also suffer from overspecialization (see subsection 1.2.3), meaning they may fail to recommend diverse or serendipitous items that are different from what the user has seen before.

2. They require a lot of domain knowledge and feature engineering to design and implement effective content analyzers and profile learners. They may also face challenges in dealing with dynamic or unstructured text, images, videos, audio, etc.

1.2.1.2 Collaborative Filtering

Collaborative filtering uses the ratings or feedback of other users who have similar tastes or preferences to a given user, and recommends items that those similar users liked or rated highly [10, 13]. To illustrate, if you and another user both enjoyed watching the same movies, then collaborative filtering would suggest other items that the other user liked but you haven't seen yet (rightmost side of figure 1.2).

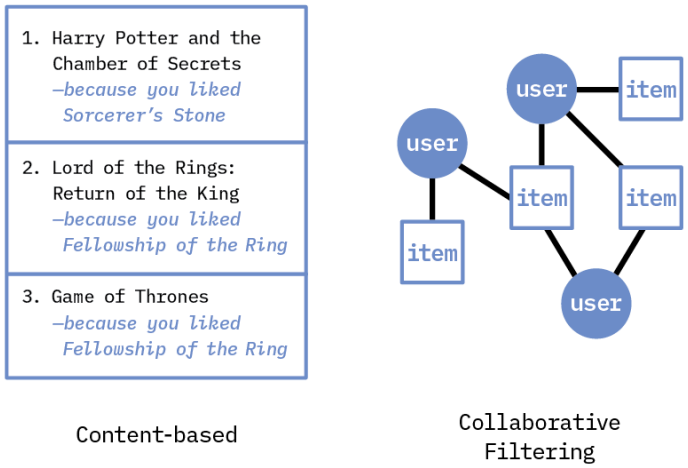

Figure 1.2: Content-based vs Collaborative filtering approaches [14]

Matrix Factorization Models. Matrix factorization models are a class of model-based collaborative filtering algorithms that work by decomposing the user-item interaction matrix into the product of two lower dimensionality matrices. These matrices represent latent features or factors that capture the preferences of users and the characteristics of items [15].

Matrix factorization models can be formulated as an optimization problem, where the objective is to minimize the reconstruction error between the original matrix and the product of the latent feature matrices, subject to some regularization terms. The optimization problem can be solved by various methods like the gradient descent algorithm [15].

Matrix factorization models[1] have several advantages over memory-based (see subsection 1.2.1.2) collaborative filtering techniques [10, 15]:

- They can handle sparse data more effectively by filling in missing values with reasonable estimates based on latent features.
- They can reduce dimensionality and noise by extracting only relevant features from high-dimensional data.
- They can improve scalability and performance by using efficient numerical methods to solve large-scale problems.

Some matrix factorization models are: singular value decomposition (SVD), non-negative matrix factorization (NMF) [13], probabilistic matrix factorization (PMF) [15].

Neighborhood Models. The neighborhood approach[2] is a common way to make an RS based on similarities between users or items [15]. It follows three main steps: (1) making the data more

[1] Also referred to as *model-based* [13].
[2] Also referred to as *memory-based* [13].

consistent by removing biases, (2) finding the most relevant and reliable neighbors for each user or item, and (3) combining the ratings of the neighbors to predict unknown ratings [10, 15].

The most relied-on method traditional RSs apply is the rating matrix. The rating matrix is a grid that shows how users rate items, such as movies, books, or products. A rating matrix can be defined as $(r_{ij})_{1\leq i\leq u, 1\leq i\leq n}$, where r_{ij} is the rating of user i for item j, u is the number of users and n is the number of items. The ratings can be numbers or categories [10, 13]:

- **Continuous ratings:** which are real numbers that can take any value within a predefined range. Continuous ratings allow for fine-grained and nuanced expressions of preferences.
- **Interval-based ratings:** which are discrete numbers that belong to a finite set of intervals. Interval-based ratings are less precise than continuous ratings, but they are easier to elicit and interpret. A common example of interval-based ratings is the star rating system, where users can rate items from one to five stars.
- **Ordinal ratings:** which are rankings that indicate the relative preferences of users on a subset of items. Ordinal ratings do not provide information about the absolute or the difference in preferences, but they capture the order of preferences. Ordinal ratings are useful when users have difficulty assigning numerical values to their opinions.
- **Binary ratings:** which are boolean values that indicate whether a user likes or dislikes an item. Binary ratings are the simplest form of ratings, but they also lose a lot of information about the intensity and diversity of preferences.
- **Unary ratings:** which are positive indicators that a user has interacted with an item in some way. Unary ratings do not have negative counterparts, so they only reflect the presence of preferences, not the absence. Unary ratings are also often derived from implicit feedback, such as views or downloads.

1.2.1.3 Hybrid Recommender Systems

A hybrid filtering Recommender System is a type of RS that combines multiple recommendation techniques, such as content-based and collaborative filtering, to produce better suggestions for users. The main advantage of hybrid filtering is that it can overcome some of the limitations of single-technique systems (e.g. data sparsity, cold start, and overspecialization) [10].

A hybrid filtering recommender system can use different methods to integrate multiple techniques, such as weighted voting, feature augmentation, meta-level learning, or switching [13, 16].

1.2.2 Non-Traditional Recommender Systems

Non-traditional RSs are systems that do not necessarily rely on rating matrices to provide personalized suggestions. They leverage temporal or context-aware information to capture the dynamics and diversity of user preferences [3]. In this light, context-aware RSs use additional

information about the user's situation to adapt the recommendations to the specific context [17]. Similarly, sequential RSs model the order and patterns of user-item interactions like the browsing or purchasing history to predict the next item that the user may be interested in. These technologies aim to improve the accuracy and relevance of recommender systems by considering more factors that influence user behavior [8].

1.2.2.1 Context-Aware Recommender Systems

Context-aware recommender systems are systems that generate more relevant recommendations by adapting them to the specific contextual situation of the user. The motivation is that users' preferences and needs may vary depending on various factors such as time, location, mood, social setting, etc. These factors are called *contextual factors* and can be classified into three types: user-related, item-related, and environment-related [17].

Context Definition. A common topic in the research on context-aware systems is the definition of context. Initially, context was understood as the user's location, the nearby people and objects, and the changes in these elements. However, this definition has been expanded over time to include other factors, such as the date, the season, and the temperature. Moreover, some researchers also take into account the physical and conceptual states of interest for a user, the user's emotional state and define context as any information that can characterize and is relevant to the interaction between a user and an application [17].

There are three main approaches to incorporating contextual information into rating-based recommender systems: pre-filtering, post-filtering [17], and contextual modeling [18].

- **Pre-filtering:** selects a subset of ratings that match the current context before applying a standard recommendation algorithm.
- **Post-filtering:** adjusts the output of a standard recommendation algorithm based on the current context [17].
- **Contextual modeling:** integrates contextual information into the recommendation algorithm itself [18].

1.2.2.2 Sequential Recommender Systems

Sequential RSs are systems that suggest items that may be of interest to a user by mainly modeling the sequential dependencies over the user-item interactions. User preferences and needs may change over time and depend on their previous actions. To illustrate, if a user buys a pen, they may also buy a textbook and a ruler afterward [8].

Sequence Definition. A *sequence* is a list of user-item interactions with a clear chronological order. Each interaction consists of a user ID, an item ID, and optionally some additional information like rating, feedback, or context — creating an overlap with the previously described context-aware

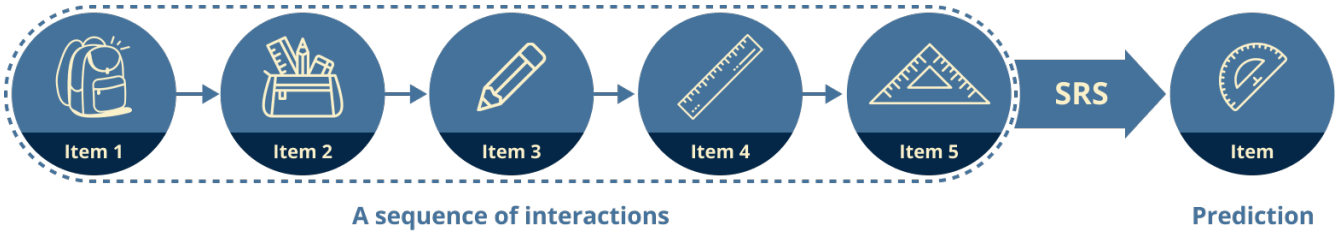

Figure 1.3: School supplies example sequence for a typical sequence-item prediction

RSs. These features can help enrich the representation of users and items and capture more complex patterns in their interactions. A sequence can be either long-term or short-term depending on how much history it covers. A long-term sequence may span over several *sessions*[3] or days, while a short-term sequence may only cover one session or a few minutes [3].

Markov Chain Models. One way sequential RSs are achieved is to use a Markov chain model which is considered to be among the more conventional sequential recommender systems' models [8]. A Markov chain is a stochastic process that describes the transitions between states based on some probabilities. In this case, a state can be an item that a user interacts with, and a transition can be a user's action that leads to another item. A sequence of items can then be represented as a Markov chain, where each item depends only on the previous one.

Formally, let S be a finite set of states, and let P be a matrix of transition probabilities, where P_{ij} is the probability of moving from state i to state j. A Markov chain model is a triplet $\{\mathbf{ST}, \mathbf{P}_t, \mathbf{P}_0\}$, where $\mathbf{ST}$ denotes the state space comprising all the unique interactions, $\mathbf{P}_t$ represents the $m \times m$ one-step transition probability matrix between m distinct interactions, and $\mathbf{P}_0$ indicates the initial probability distribution over the states in $\mathbf{ST}$. The first-order transitional probability from interaction o_i to o_j is given by [3, 13]:

$$\mathbf{P}_t(i,j) = \mathbf{P}(o_i \rightarrow o_j) = \frac{\text{freq}(o_i \rightarrow o_j)}{\sum_{o_t} \text{freq}(o_i \rightarrow o_t)}$$

Then, a transition path, for instance, $\{o_1 \rightarrow o_2 \rightarrow o_3\}$, is predicted by computing its probability using the first-order Markov chain model [3]:

$$\mathbf{P}(o_1 \rightarrow o_2 \rightarrow o_3) = \mathbf{P}(o_1) * \mathbf{P}(o_2|o_1) * \mathbf{P}(o_3|o_2)$$

The reference paths are determined by selecting the sequences of interactions with the highest probabilities. Subsequently, the item(s) that follow the sequence in the reference path is recommended.

[3] A session refers to a finite and delimited series of interactions. This concept will be elaborated in more detail in the subsection 1.3.1.

Deep Learning Models. Another recent way of designing sequential RSs involves the use of deep learning algorithms such as recurrent neural networks (RNNs) [3, 8]. This is achieved by treating the sequence of items as a sequence of inputs to an RNN, which can learn to capture the dependencies and temporal dynamics among items. The RNN then outputs a hidden state vector that represents the user's current preference, and this vector can be used to generate recommendations based on some similarity or ranking function. A naïve approach would be the following, suppose we have four items: A, B, C and D. The user's sequence of interactions is: $A \to C \to B \to D$. We can feed this sequence into an RNN,

$$\begin{aligned} h_0 &= \text{initial state} \\ h_1 &= f(h_0, A) \\ h_2 &= f(h_1, C) \\ h_3 &= f(h_2, B) \\ h_4 &= f(h_3, D) \end{aligned}$$

where f is some nonlinear activation function. The final state h_4 can then be used to compute a score for each item based on some function g,

$$s_A = g(h_4, A) \qquad s_B = g(h_4, B) \qquad s_C = g(h_4, C) \qquad s_D = g(h_4, D)$$

The scores are then used to rank the items and recommend the top ones to the user.

1.2.3 Challenges and Obstacles

Recommender systems, both conventional and novel, encounter various challenges that hinder their effectiveness. The main sources of these challenges are the scarcity or low quality of information.

Lack of User Activity. A recommender system depends on the activities performed by multiple users. It studies different users' behavior and then lists the products or services according to their activity and feedback. If there is not enough user activity, it becomes difficult to generate accurate recommendations.

Lack of Data. Much like the lack of user activity, the availability of abundant data is also needed by recommender systems. Data sparsity occurs when there are many items but only a few ratings per item. This makes it hard to find similar users or items based on ratings.

New Item Introduction. The cold start problem refers to the difficulty of recommending new items that have no ratings or feedback from users. A recommender system may not be able to suggest these items to potential users who might like them [10].

Scalability. As the number of users and items grows, a recommender system needs to handle large amounts of data and computations efficiently. It also needs to adapt to changing user preferences and item features over time [10].

Diversity. A recommender system may tend to suggest items that are similar to what a user has already liked or purchased, resulting in a lack of diversity in recommendations. This may reduce user satisfaction and exploration. This is referred to as overspecialization [10].

Privacy. A recommender system collects personal information from users such as their ratings, preferences, browsing history, etc. This raises concerns about how this data is stored, processed and shared with third parties. Users may not want their personal data to be exposed or misused by others [10].

For the case of sequential recommender systems, they face some additional challenges:

Learning Higher-order Sequential Dependencies. Higher-order sequential dependencies refer to complex patterns in user-item interaction sequences that cannot be captured by simple Markov models. For example, a user may buy a book after browsing several related books in different genres or topics. The system needs to learn these higher-order dependencies from long sequences of data [8].

Learning Long-term Sequential Dependencies. Long-term sequential dependencies refer to influences from distant past events on current decisions. A user may buy a product after being exposed to an advertisement several days ago. Sequential recommender systems need to learn these long-term dependencies from sparse and noisy data [8].

Learning Sequential Dependencies Over Sequences With Noise. A major challenge in sequential recommendation systems is to deal with the uncertainty of user shopping behaviors, which often results in noisy and irrelevant interactions in the user-item interaction sequences. These interactions can interfere with the prediction of the next interaction. In reality, not all historical interactions have the same relevance to the next interaction. Some may have a strong sequential dependency, while others may have a weak or no dependency[4]. For instance, in a shopping sequence {ruler, egg, backpack, pen}, the item "egg" may be an outlier that does not correlate with the other items or the next item, which could be a protractor with high probability. The next item prediction should depend on ruler, backpack and pen more than on the egg [8].

1.2.4 Evaluation

Recommender systems aim to provide personalized suggestions to users based on their preferences and needs. Evaluating recommender systems involves measuring how well they achieve this goal using different metrics and methods.

1.2.4.1 Metrics

Some common metrics for evaluating RSs are:

[4]One of the existing solutions to this challenge is the use of attention models.

1. **Accuracy:** is how close the recommendations are to the actual preferences of the users. This can be measured by comparing the predicted ratings or rankings with the ground truth ratings or rankings, if available. Some examples of accuracy metrics are:
   - Precision: is the fraction of relevant items among the recommended items. It measures how precise or exact the system is in providing useful recommendations [10].
   - Recall: is the fraction of relevant items that are recommended by the system out of all relevant items. In other words, how complete or exhaustive the system is in covering user interests [10].
   - Mean reciprocal rank (MRR): is the average of the reciprocal ranks of the first relevant item in each list of recommendations. It measures how quickly or early the system can provide a useful recommendation [5].
2. **Coverage:** is how many items or users can be recommended by the system. This can be measured by calculating the percentage of items or users that receive at least one recommendation. Coverage reflects how broad or diverse the system is in providing recommendations [10].
3. **Novelty:** is how surprising or unexpected the recommendations are for the users. This can be measured by calculating the average popularity or familiarity of the recommended items among the users. Novelty reflects how creative or innovative the system is in providing recommendations [10].

1.2.4.2 Methods

Some common methods for evaluating recommender systems are:

1. **Offline Testing:** using historical data to simulate user feedback and compare different algorithms or parameters. This method is fast and cheap, but it may not reflect the real behavior of users or capture dynamic changes in their preferences [15].
2. **Online Testing:** deploying different versions of the system to real users and measuring their responses. This method is more realistic and reliable, but it may be costly, risky, and unethical [15].
3. **User Studies:** conducting surveys or interviews with a sample of users to collect their opinions and feedback on the system. This method can provide rich and qualitative insights, but it may suffer from low response rates, biases, and scalability issues [15].

1.3 Session-Based Recommender Systems

While the previous Recommender Systems do a decent job suggesting recommendations for a more general recommendation task, their approach implies that the user's past interactions have a uniform influence on his present preference regardless of their temporal distance or relevance. These

assumptions may not hold true in real-world scenarios for two main reasons: (1) A user's selection of items is influenced not only by his long-term historical preference, but also by his short-term recent preference and the time-sensitive context. This short-term preference is reflected in the user's most recent interactions, which often constitute a small fraction of the overall historical interactions; (2) A user's preference towards items is likely to change over time rather than remain constant, that is, it has temporal dynamics (see figure 1.4).

One of the challenges of sequential RSs is to define the boundaries of a sequence. A sequence can be long and contain multiple subsequences that are not contiguous in time. A sequence does not have a clear criterion for delimitation or segmentation, which makes it difficult to capture temporal dependencies. To address this problem, Session-based RSs — which are a subtype of sequential RSs — propose to use sessions as the basic unit of analysis. A session is a coherent and relatively short sequence of user actions that reflects a specific goal or intent [3, 19]. Session-based RS can leverage the session information to provide more accurate and personalized recommendations.

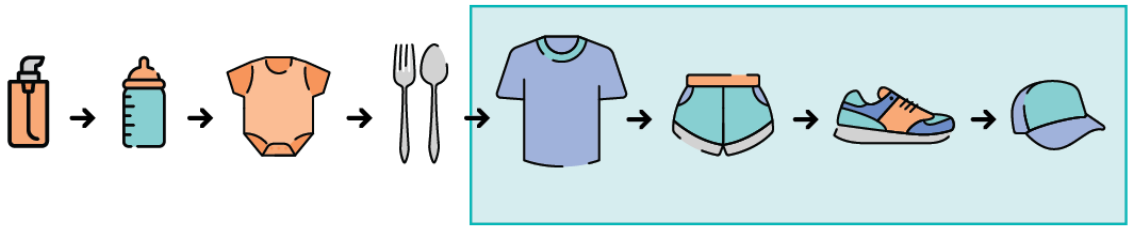

Figure 1.4: Although an interaction sequence may appear homogeneous at first glance, intricate differences may hint for a change of user interest [14]

1.3.1 Session Definition

A session is a collection of interactions that has a clear boundary, which defines the start and end of a specific session. The interactions within a session can be either ordered or unordered, depending on whether they follow a chronological order or not. A user's session data typically comprises multiple sessions that occur at different times and are separated by various boundaries with unequal time intervals between them [3]. On the other hand, as stated previously, a sequence is simply an ordered list of historical interactions that have a clear order. A user's sequence data usually contains a single sequence with one boundary for it. The timestamps in most sequence data are only used to sort the elements within a sequence, and no explicit time intervals are included or considered [8].

A boundary implies that the interactions or elements within it have co-occurrence-based dependencies[5], meaning that they tend to occur together. Co-occurrence-based dependencies are the basis of SBRSs, especially for those that use unordered session data. Order indicates that

[5]Co-occurrence-based dependencies refer to the connection between two or more phenomena, objects, or events, which have a tendency to happen together.

the interactions or elements within a session or a sequence have sequential dependencies, meaning that they influence each other in a temporal order [3].

1.3.2 Framework

Researchers have proposed various approaches to SBRSs, which can be classified into three sub-areas according to their recommendation tasks. This classification framework aims to clarify the differences and similarities among the existing methods and address the issues of inconsistency and confusion in the literature [3]. The three sub-areas are: *next interaction recommendation*, *next partial-session recommendation*, and *next session recommendation*.

- **Next Interaction Recommendation:** Next interaction recommendation focuses on predicting the most likely interaction that a user will perform in the current session, given the previous interactions within it. For example, it can recommend the next product that a user will click or buy based on the intra-session dependencies.

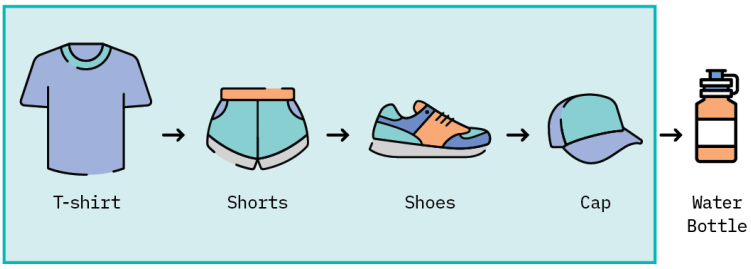

Figure 1.5: Next interaction recommendation focuses on recommending one item at a time [14]

- **Next Partial-session Recommendation:** Next partial-session recommendation tries to predict all the remaining interactions that a user will perform to complete the current session, given the previous interactions within the same session. For instance, it can recommend all the products that a user will buy to fill a basket based on the intra-session dependencies. This sub-area is more realistic and practical than the previous one, as users often interact with multiple items in a session rather than one [3].
- **Next session recommendation:** Next session recommendation aims to predict the entire next session that a user will initiate, given the historical sessions of the user. It can recommend the next basket that a user will buy based on the inter-session dependencies. Sometimes, inter-session dependencies are also used to enhance the performance of the first two sub-areas.

1.3.3 Approaches

Wang *et al.*'s SBRS survey [3] identifies three super-classes of SBRS approaches according to their technical perspective and the methods involved: conventional approaches, latent representation approaches, deep neural network approaches (see figure 1.6).

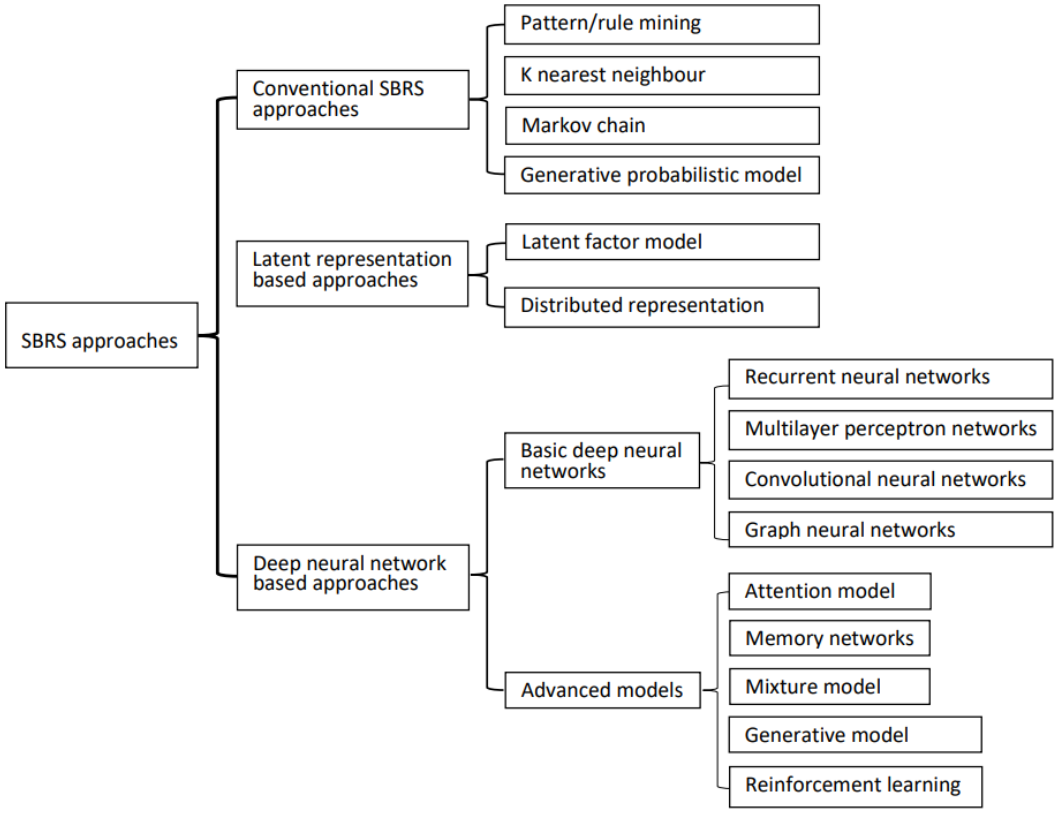

Figure 1.6: The categorization of SBRS approaches [3]

1.3.3.1 Conventional Approaches

Conventional methods for SBRS employ traditional data mining or machine learning techniques to extract the latent dependencies from session data. There are four classes [3]:

Pattern/Rule Mining. Pattern/Rule Mining is a common approach to address the SBRS challenge, leveraging the patterns or rules that can be extracted from historical sessions to infer the user's current interests. Pattern/rule mining based SBRS can be divided into two categories: *frequent pattern/association rule mining based SBRS* and *sequential pattern mining based SBRS*.

1. **Frequent pattern/association rule mining:** mines the association rules between different items that co-occur frequently within unordered sessions, and uses them to rank the candidate items for recommendation. Let A and B be two items, if many users have interacted with items A and B in the same session regardless of order, then the rule $A \rightarrow B$ can be derived, indicating that users who interact with A are likely to interact with B as well. There are various algorithms for frequent pattern mining, such as FP-Tree [20], which can efficiently extract the frequent patterns from a large session set S over an item set V [3].

2. **Sequential pattern mining based:** mines the sequential patterns between items that appear in a certain order within ordered sessions, and uses them to predict the next item for recommendation. For example, if many users have interacted with items A, B and C in a

sequential order in different sessions, then the pattern $A \rightarrow B \rightarrow C$ can be derived, indicating that users who interact with A and B are likely to interact with C next [3, 21].

K Nearest neighbor. The basic idea of the K Nearest neighbor (KNN) approach is to measure the similarity between the current session and the previous sessions in the data, and then select the K most similar item (or sessions) as neighbors. The similarity can be computed using different metrics (cosine similarity, Jaccard similarity, Pearson correlation, etc.) Based on the neighbors, a score is assigned to each candidate item that reflects its relevance to the current session. The score can be calculated as a weighted sum of the similarities between the current session and the neighbors that contain the item. The items with the highest scores are then recommended to the user. There are two main variants of the KNN approach for SBRS: item-KNN and session-KNN [3].

- **The item-KNN approach:** it treats each item as an interaction and computes the similarity between items based on their co-occurrence in sessions. The score of a candidate item is then computed as a weighted sum of the similarities between the item and the items in the current session. Each item is represented by a binary vector, where the presence (value of "1") or absence (value of "0") of the item in a given session is indicated by each element. Thus, the similarity measure of choice can be applied to their vectors to compute the similarity between items [3].
- **The session-KNN approach:** instead of the former, it considers each session as an interaction and computes the similarity between sessions based on their common items. The score of a candidate item is then computed as a weighted sum of the similarities between the current session and the neighbors that contain the item [3].

Markov Chain Models. As mentioned previously, a Markov chain is a stochastic process that models the transitions between states with certain probabilities. In SBRSs, the states can be either items or intentions, where an intention is a tuple of category and action that represents the user's underlying goal or interest. For example, an intention can be **(book, buy)** or **(movie, rate)**. The transition probabilities can be estimated from the observed data or learned from latent factors. The basic idea of Markov chain models for SBRSs is to use the previous interactions or intentions in a session to predict the next one, based on the assumption that the user's behavior follows a Markov property, i.e., the future state depends only on the current state or a fixed number of previous states.

Latent Markov Embedding (LME) based models for SBRSs, different from the basic Markov based models (cf. paragraph 1.2.2.2) propose to embed the items and the Markov chains into a low-dimensional Euclidean space, where the transition probabilities are computed based on the distance between the embeddings. This way, LME models can infer the latent similarities between items and leverage them to make more accurate and diverse recommendations. The objective of LME models is to solve the data sparsity issue in limited observed data [3].

Generative Probabilistic Models. The general idea of generative probabilistic model based methods is to first discover the hidden categories (e.g. topics or genres) of items[6] in sessions and then model the patterns of change among these hidden categories within or across sessions. Next, they estimate the hidden category of the next item based on the modeled patterns. Lastly, they predict specific items as the next item depending on the estimated hidden category of items. Commonly, the latent topic model is employed to discover the hidden categories and the patterns of change among them [3].

1.3.3.2 Latent Representation Approaches

Latent representation approaches for SBRSs use shallow models to map each interaction to a low-dimensional latent vector that summarizes its information and dependencies with other interactions. The latent vectors are then used to predict the next items that the user may be interested in. There are two main types of latent representation approaches for SBRSs: latent factor model based SBRSs and distributed representation SBRSs.

Latent factor model based SBRSs. They assume that each item and each interaction can be represented by a latent factor vector, and the relevance between an item and an interaction is measured by the inner product of their vectors. These approaches use matrix factorization or tensor factorization techniques[7] to learn the latent factors from the observed interactions [3]. One example of this approach is Factorizing Personalized Markov Chains (FPMC) [22].

Distributed representation SBRSs. Shallow neural networks[8] are used to learn distributed representations for items and interactions. These approaches use various network architectures and even attention mechanisms to encode the sequential patterns and temporal dynamics of the interactions. The distributed representations are then fed into a prediction layer to generate recommendations [3].

1.3.3.3 Deep Neural Network Approaches

RNN-based approaches. Also stated previously, they use Recurrent Neural Networks (RNNs) to model the sequential nature of user behavior and capture the temporal dependencies among items. RNN-based approaches can handle variable-length input sessions and learn long-term dependencies (see 1.2.2.2). Conventionally, other types of RNN cells are used for this task like LSTMs [23] and GRUs [24] to reduce problems faced by RNNs in practice, which we will lay out in the next chapter [3].

MLP-based approaches. As the name implies, Multi-layer Perceptrons (MLPs) are used to learn the nonlinear mapping between user features and item features. MLP-based approaches can

[6] Like music genres are for songs for example.
[7] e.g. Tucker Decomposition.
[8] Only 1 or 2 hidden layers are present in shallow neural networks.

capture complex interactions between users and items, this approach is especially useful if the sessions are unordered [3].

CNN-based approaches. They use Convolutional Neural Networks (CNNs) to extract local and global features from user behavior and item attributes. CNN-based approaches can exploit the spatial structure of the input data and reduce the number of parameters. Essentially, a CNN-based SBRS initially uses filtering and pooling operations to develop a better understanding of the context of each session, and then applies this knowledge to make appropriate recommendations. They are a favorable option for SBRSs because they offer two distinct advantages [3]:

1. Firstly, they are able to relax the rigid assumption regarding the order in which interactions take place within sessions, thereby enabling the model to be more robust [3].
2. Secondly, CNNs are particularly adept at identifying local features from specific areas and establishing relationships between different areas within a session. This allows them to effectively capture the collective dependencies present in session data at the union-level [25].

GNN-based approaches. Graph neural networks are used to model the user-item interactions as a bipartite graph and propagate information along the graph edges. GNNs have demonstrated considerable expressive power in modeling the intricate relationships inherent in graph-structured data by incorporating Deep Neural Networks. To leverage this capability, some researchers have introduced GNN to model the complex transitions within or between sessions and develop more effective SBRSs [3].

1. Initially, sessions are transformed into a graph/graphs by mapping each session onto a chain. In this context, each interaction within a session is regarded as a node in the corresponding chain, while an edge is established between each pair of adjacent interactions in the session.
2. Following this, the constructed graph is introduced to a GNN, which generates an informative embedding for each node (interaction) by encoding the complex transitions over the graph into the embeddings. Depending on the specific architecture of the GNN model, GNN approaches for SBRSs can be generally categorized into three groups: *Gated Graph Neural Networks* (GGNN), *Graph Convolutional Networks* (GCN), and *Graph Attention Networks* (GAT) [3].
3. Finally, these embeddings are employed in the prediction module for session-based recommendations.

Advanced Models. Advanced approaches include *attention models*, *mixture models*, *generative models* and *reinforcement learning*. These methods either use a more advanced way to make recommendations or combine some of the methods we mentioned before. Notably:

- **Attention models:** are designed to assign different weights to different items or features in a session, based on their relevance to the user's current interest[9]. By doing so, they can focus on

[9]We will come back to the idea of attention — soft-attention specifically — and how it is achieved in the next

the most important information and filter out the noise or redundancy in a session. Attention models can be applied to different basic approaches for SBRSs like distributed representation learning, RNN, or GNN [3]. For instance, Li *et al.* [26] proposed an attention-based RNN model that uses an attention mechanism to learn the importance of each item in a session and generate more accurate recommendations.

- **Mixture models:** can model the heterogeneity and diversity of user preferences in a session by relying on multiple sub-models, each of which specializes in capturing latent subgroups or topics of items in a session and generate personalized recommendations for each subgroup or topic [3].
- **Reinforcement learning:** is an advanced algorithm that can optimize the long-term reward of SBRSs. Reinforcement learning can learn from the feedback of users and dynamically adjust the recommendation strategy according to the user's feedback extracted from the *conversation* he has with the recommender system. This conversation consists of the RS suggesting an item and the user giving some feedback on it, and then repeating this process with another item [3].

1.3.4 Future aspects and outlook

Although SBRSs already come with advantages when compared to traditional RSs, they still face some challenges which, when surmounted may greatly improve the quality of the recommendations. Promising research directions for further development of Session-Based Recommendation Systems (SBRS) are for instance [3]:

1. Incorporating users' explicit general preferences into SBRS [3]:
   (a) Investigating more advanced approaches to learn users' general preferences from explicit preference data.
   (b) Exploring ways to combine both users' long-term general preferences and short-term preferences together when ranking the candidate items in SBRS.
2. Incorporating users' implicit general preferences into SBRS [3]:
   (a) Developing novel approaches to learn users' implicit general preferences from their transaction behavior data, which includes view, click, add to cart, and purchase, in the absence of explicit feedback.
   (b) Simultaneously learning a user's implicit general preference and short-term preference and effectively integrating them for accurate recommendations.
3. Domain-specific contextual information [3]:

chapter.

(a) Contextual information can be domain-specific, and therefore, incorporating domain-specific contextual information can be a promising direction for future research.

(b) Different users may respond differently to the same contextual factor, and therefore, personalized contextualization can be a promising direction for future research. Personalizing contextualization can involve understanding how different users perceive the same contextual factors and incorporating that knowledge into the recommendation system.

1.4 Conclusion

Session-based Recommender Systems have demonstrated their ability to offer more accurate and relevant recommendations in comparison to traditional recommendation systems. This is due to their ability to consider the user's current context, preferences, and behavior in real-time, rather than solely relying on historical data. Session-based Recommender Systems also address the cold-start problem by providing recommendations for new or infrequent users who lack historical data. These systems provide a more personalized and engaging user experience, which can increase user satisfaction, loyalty, and ultimately, business revenue.

To enhance Session-Based Recommender Systems and recommendation tasks in general, Graph Neural Networks (GNNs) offer a promising solution to deal with complex, graph-structured data. Gated Graph Neural Networks (GGNNs) have demonstrated their ability to effectively capture the structural information in such data and generate accurate recommendations. In the following chapters, we will delve into the intricacies of how GGNNs process graph-structured data and examine their potential applications in the field of recommendation systems.

CHAPTER TWO:

Gated Graph Neural Networks

2 Gated Graph Neural Networks

2.1 Introduction

Graphs are powerful representations that capture intricate relationships and structures in a wide range of domains, including social networks, molecular chemistry, recommendation systems, and more. Analyzing and inferring properties from graphs have become increasingly important in various fields, and Gated Graph Neural Networks (GGNNs) [27] have emerged as an intriguing technique for tackling these tasks. GGNNs leverage gated mechanisms to propagate messages between nodes, allowing for information exchange and aggregation throughout the graph. By capturing both local and global dependencies, GGNNs enable effective modeling of complex relationships within graph structures.

In this chapter, we delve into the fascinating world of GNNs, exploring their applications and theoretical foundations. We begin by discussing the broader context of artificial intelligence and its connection to graph analysis. Understanding how GNNs fit within the realm of AI provides a holistic perspective on the significance and potential of these networks. We explore the underlying concepts and mathematical formalisms that enable GGNNs to capture and leverage the rich information contained within graphs.

We delve into the principles and applications of attention mechanisms, highlighting their synergy with GGNNs and how they enhance the network's ability to extract meaningful insights from graphs.

2.2 Machine Learning

Machine Learning (ML) is a subfield of artificial intelligence (see figure 2.1) that enables machines to learn from data without being explicitly programmed. In other words, instead of writing a program that performs a specific task, a machine learning algorithm can be trained on a dataset to learn patterns and relationships in the data, and then use this knowledge to make predictions or decisions on new, unseen data.

Machine learning has been applied to a wide range of domains, including image and speech recognition, natural language processing, autonomous driving, and healthcare. As such, it has become an increasingly important tool for solving complex problems and creating intelligent systems.

2.2.1 Artificial Intelligence

Artificial Intelligence (AI) is the emulation of human intelligence in machines, allowing them to perform tasks that previously required human intervention. Through AI, machines can learn from experience, recognize patterns, and adapt to new situations, making them valuable for diverse applications, from robotics to healthcare to finance. AI technologies encompass a variety of fields,

including machine learning, deep learning, natural language processing, robotics, and expert systems, among others [28].

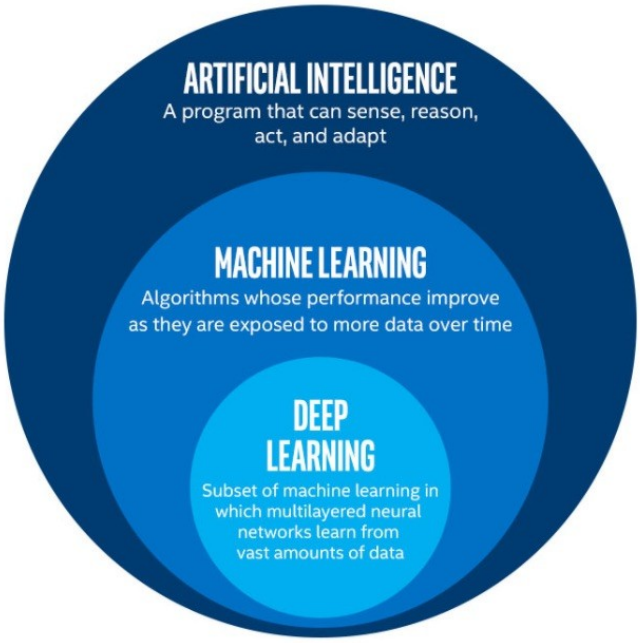

Figure 2.1: The relationship between Artificial Intelligence, Machine Learning, and Deep Learning

2.2.2 Basics of Machine Learning

Specifically, machine learning refers to the development of algorithms that can automatically improve their performance on a particular task over time by learning from data or experience. The goal of machine learning is to enable computers to learn and make decisions or predictions without being explicitly programmed to do so [29]. The experience that a machine learning algorithm learns from can come in various forms: labeled or unlabeled data, feedback from users, or interactions with the environment. The class of tasks that machine learning algorithms can be applied to is quite broad, including tasks like *classification*, *regression*, *clustering* [29].

Machine learning methods are typically divided into three categories – sometimes four if you include Semi-Supervised Learning – based on the task and the type of feedback available to the learning system (see figure 2.2):

- **Supervised Learning:** the algorithm is presented with a training dataset consisting of input/output pairs, where the input features are typically represented as a vector, and the output labels are some predefined categories or values. The goal of the algorithm is to learn a function that can accurately predict the output label given a new input[10] [30].
- **Unsupervised Learning:** opposite to supervised learning, unsupervised learning learns patterns in the data without any explicit supervision or guidance from labeled examples. The

[10]e.g. Image classification, speech recognition, natural language processing, and recommendation systems, etc.

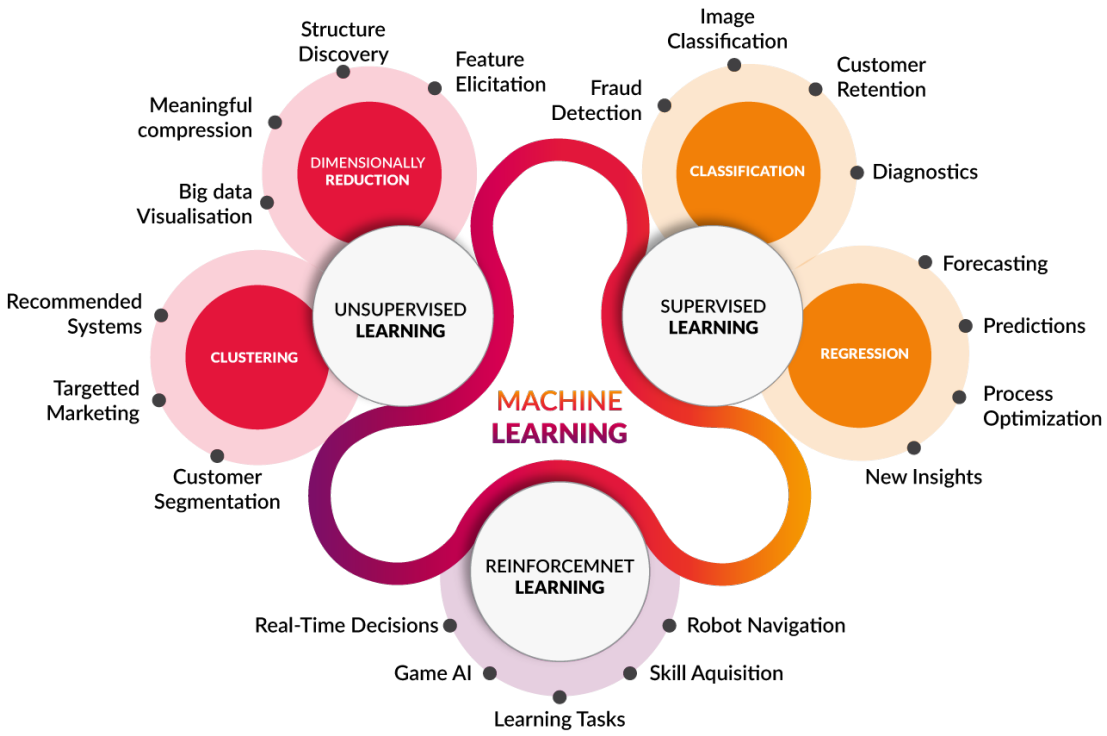

Figure 2.2: Machine learning types and their applications / Christelle Julias, smartpredict.ai

algorithm is presented with a dataset consisting of input features but no corresponding output labels [29, 30]. The goal of the algorithm is to discover meaningful patterns or structures in the data, like clusters, principal components, or frequent patterns[11].

- **Semi-Supervised Learning:** the algorithm is given both labeled and unlabeled examples, with the assumption that the labeled examples are more expensive or difficult to obtain than the unlabeled ones. The goal is to use the labeled examples to guide the learning process and improve the accuracy of the model on the unlabeled data [30].
- **Reinforcement Learning:** the agent[12] learns to take actions in an environment to maximize a cumulative reward signal. The agent receives feedback in the form of a reward or penalty based on its actions, and the goal is to learn a policy that maximizes the expected cumulative reward over time [29, 30].

2.2.3 Tasks

Machine learning allows us to address complex tasks that cannot be solved using fixed programs created and developed by humans [30]. To keep things simple, we will only introduce regression, classification and clustering since they are the most common and versatile techniques (see figure

[11] e.g. Anomaly detection, data compression, and data visualization, etc.

[12] An agent is an entity that perceives its environment through sensors and acts upon the environment through actuators to achieve specific goals [28].

2.3):

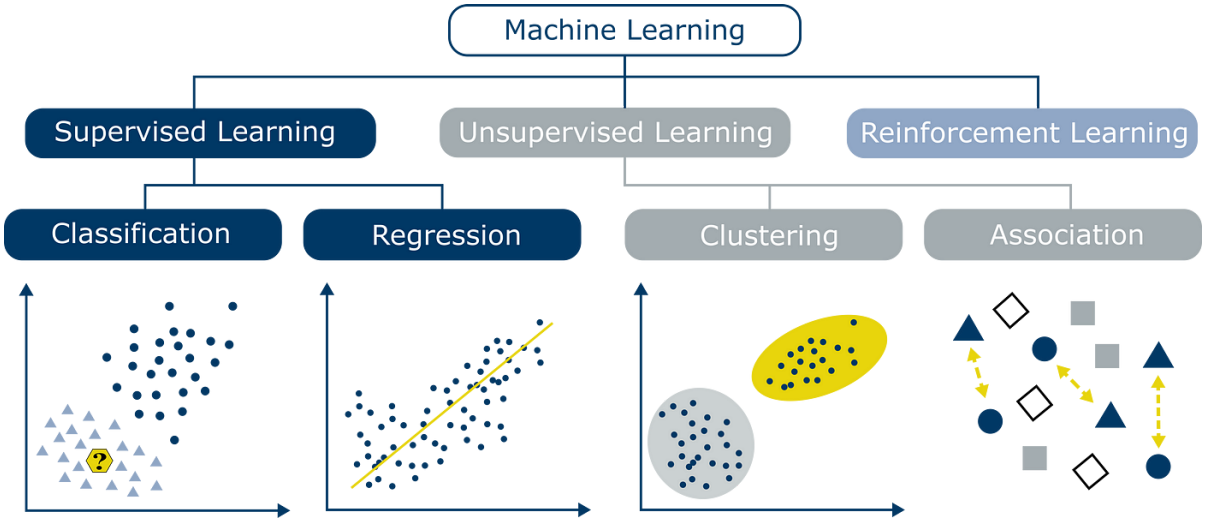

Figure 2.3: Most common machine learning tasks / Dominik Polzer, TowardsDataScience

- **Regression:** the computer is given a set of input data and tasked with predicting a numerical value, i.e, a function f that maps the input data to a numerical output[13] $f : \mathbb{R}^n \to \mathbb{R}$. A regression task could be predicting the price of a house based on its characteristics such as size, number of bedrooms or location [30].
- **Classification:** the goal is to predict the class or category that a new piece of data belongs to, based on its features or attributes [30]. The learning algorithm is trained on a set of labeled examples, and the output is a function that maps the input data to a discrete output (i.e. label, category). For example, identifying whether an email is spam or not based on its content and metadata. The fact that this problem has two classes signifies that it is a binary classification [29].
- **Clustering:** involves grouping similar data points together based on their features or attributes. The goal of clustering is to partition a set of data points into distinct clusters or groups, where the points within each group are more similar to each other than to those in other groups. The learning algorithm is not given labeled examples, but instead has to discover the underlying structure of the data on its own [29].

2.2.4 Limitations

Despite the many advances made in recent years, there are still several challenges that need to be addressed to make machine learning more effective and trustworthy. Addressing these challenges will be critical to the continued development and adoption of machine learning in a wide range of fields and industries. To mention a few:

[13] Here, n is the dimensionality of the feature vector.

- **Bias and fairness:** is a common limitation in machine learning, stemming from the fact that algorithms are only as objective as the data they are trained on. It can arise when the training data contains systematic errors or is not representative of the real-world population, leading to models that make incorrect or unfair predictions for certain groups [31].
- **Data quality and availability:** in many cases data can be missing or incomplete, or contain outliers that can affect the quality of the predictions. And like we have seen before, it may also be biased or unrepresentative, leading to models that perform poorly on new or unseen data. Addressing these challenges requires careful data cleaning and preprocessing, as well as strategies for handling missing or incomplete data [32].
- **Interpretability and Explainability:** interpretability refers to the ability to understand and explain how a model arrives at its predictions or decisions, while explainability refers to the ability to provide a clear and intuitive explanation for those predictions or decisions [33]. Some machine learning models are highly complex and opaque, making it difficult for humans to understand the reasoning behind their outputs. This can raise serious security concerns for fields where machine learning is asked to assist in critical operations like healthcare.

2.3 Deep Learning

Deep Learning (DL) refers to a powerful machine learning framework that involves training artificial neural networks to perform complex tasks, particularly in supervised learning [30]. The deep neural network architecture involves adding numerous layers and units, which increases its ability to represent highly complex functions. Essentially, most tasks that a human can perform quickly and efficiently can be accomplished through deep learning, as long as there is a sufficient amount of labeled training data and large models to handle the task at hand [30]. While several approaches and goals of deep learning are still in development, modern deep learning has become a potent tool for achieving a range of objectives in artificial intelligence [29].

2.3.1 Neural Networks

A Neural Network (NN) is a type of machine learning model that is better visualized as interconnected layers of nodes (see figure 2.4), each of which applies a nonlinear transformation to its inputs (see figure 2.5) [30]. The **activation function** of each node determines the output signal that is **propagated** to the next layer.

The network is trained using a **loss function** that measures the difference between its predictions and the true values. The optimization of the network's parameters, or weights, is achieved through a process called **backpropagation** [34], where the gradient of the **loss function** with respect to the weights is computed and used to update them. The propagation of signals and backpropagation of errors allow the network to learn and adjust its weights to better fit the training data [30, 34].

The architecture of the network, including the number and size of its layers, the choice of activation

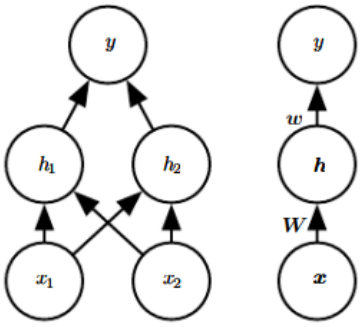

Figure 2.4: Two different ways of illustrating Neural Networks [30]

functions, and the type of loss function, can be designed to suit the specific requirements of the task at hand.

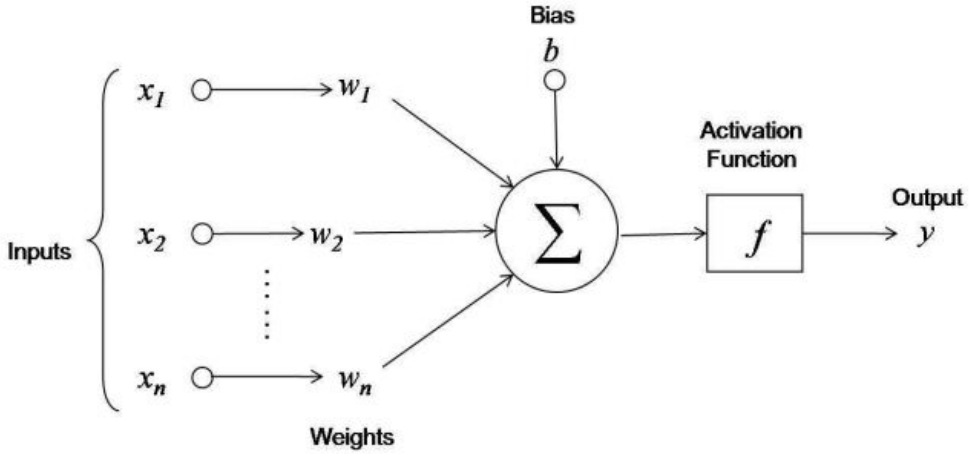

Figure 2.5: The activation of a single *neuron/perceptron* in a network

2.3.2 Learning and Training

The process of learning in neural networks involves several steps [30]:

1. **During forward propagation:** input data is fed into the network, which applies weights and biases to generate predictions.
2. **During backpropagation:** the network calculates the loss (i.e difference) between its predictions and the true values and adjusts the weights and biases accordingly. This is done through optimization techniques like gradient descent, which iteratively updates the weights and biases to minimize the error.
3. **To prevent overfitting and improve generalization:** regularization techniques like L1 or L2 regularization are applied to penalize large weights and biases.
4. By repeating this process on a large dataset, neural networks can learn to accurately predict outputs for new inputs.

Forward Propagation

Forward propagation refers to the process by which information flows forward through a feedforward neural network, starting from the input x and passing through the hidden layers, and producing an output $\hat{y}$. During training, forward propagation produces a scalar loss $L(y)$ [30].

Essentially, at each layer, the activations of the previous layer are combined linearly with learnable weights $\mathbf{W}^{(i)}$ and biases $\mathbf{b}^{(i)}$, and then transformed by a non-linear activation function $f^{(i)}$ [30]. Mathematically, in Eq. 2.1, this can be expressed as a dot product between the input activations and a weight matrix, followed by the addition of a bias vector and the application of the activation function as can be seen in figure 2.5. A layer is defined as:

$$h^{(i)} = f^{(i)}(\mathbf{W}^{(i)\top}\mathbf{X} + \mathbf{b}^{(i)}) \tag{2.1}$$

For example, the content of the input matrix $\mathbf{X}$, weight matrix $\mathbf{W}$ and bias matrix $\mathbf{b}$ can be:

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 & b_2 \end{bmatrix} \tag{2.2}$$

The shape of the input matrix $\mathbf{X}$ is $(1, 3)$, the shape of the weight matrix $\mathbf{W}$ is $(3, 2)$ and the shape of the bias matrix $\mathbf{b}$ is $(1, 2)$.

Although the analogy of a perceptron is often used to explain this process, where each neuron is connected to its previous and next layers via synapses, the underlying operation is simply a dot product between the input activations and the weight kernel, where each element of the input corresponds to a row in the matrix and each element of the weight corresponds to a column [30].

Backward Propagation

Backpropagation is a type of supervised learning algorithm that uses gradient descent to adjust the weights and biases of a neural network, it is essentially the one responsible of the learning.

The goal of backpropagation is to minimize a loss function, which measures the difference between the predicted output of the network and the true output. To do this, backpropagation computes the gradient of the loss function with respect to the learnable weights and biases of the network. The gradient is then used to adjust the weights and biases in the direction that minimizes the loss function [34].

The key to computing the gradient efficiently is the chain rule [34]. The chain rule (Eq. 2.3) allows us to compute the derivative of a composite function by multiplying the derivatives of its individual components. In our case, the composite function is the output of the network, which is a function of the weights and biases at each layer. By applying the chain rule, we can compute the gradient of the loss function with respect to the weights and biases at each layer, starting from the

output layer and working backwards towards the input layer, this is why the algorithm is called "backpropagation" [30].

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.3}$$

Once we have computed the gradient of the loss function with respect to the weights and biases, we can adjust them using gradient descent. The equation for updating the weights and biases using gradient descent is:

$$w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} - \alpha \frac{\partial L(w^{(k)})}{\partial w_{ij}^{(k)}} \tag{2.4}$$

where $w_{i,j}$ is the weight between neuron i in the previous layer and neuron j in the current layer, α is the learning rate, and $\frac{\partial L(w^{(k)})}{\partial w_{i,j}}$ is the partial derivative of the loss function with respect to the weight $w_{i,j}$. This equation is applied to every weight in the network to update its value in the direction of the negative gradient.

Optimization

The selection of an optimization algorithm is a crucial step in optimizing a neural network [30]. In machine learning, there are three main types of optimization methods: batch, stochastic, and a combination of both, known as minibatch[14] methods [35].

- **Stochastic Gradient Descent (SGD):** is a simple optimization algorithm that updates the model's parameters in the direction of the negative gradient of the loss function [30].
- **Root Mean Squared Propagation (RMSProp):** is an optimization algorithm that uses a moving average of squared gradients to scale the learning rate [30].
- **Adaptive Moment Estimation (ADAM):** is an optimization algorithm that combines the benefits of both RMSprop and momentum optimization[15] by using a moving average of both the gradients and their squares [30].

Regularization

Regularization is an important technique used in deep neural networks to prevent overfitting [30, 36], which occurs when a model becomes too complex and starts to memorize the training data instead of generalizing to new data [30].

[14] Minibatch methods use a portion of the training set during each epoch, which accelerates neural network training and allows for an unbiased estimate of the expected gradient. In deep learning, minibatch methods are often preferred due to their efficiency and unbiased estimates.

[15] Momentum optimization is a technique used in gradient-based optimization algorithms for accelerating the convergence of the optimization process. It is based on the idea of accumulating a moving average of past gradients to update the parameters of the model [35].

Regularization is classified into two types [30]:

- **Explicit:** is achieved by adding a term to the optimization problem to impose a cost on the optimization function, which makes the optimal solution unique.
- **Implicit:** includes early stopping and using a robust loss function to prevent overfitting.

L1 and L2 regularization are two popular types of explicit regularization used in deep neural networks [36]:

- **L1 regularization (Lasso regularization):** imposes a penalty term on the loss function proportional to the L1 norm of the weight vector. This induces sparsity in the weight vector, as many weights are shrunk to zero. This leads to a simpler and more interpretable model that avoids overfitting.
- **L2 regularization (Ridge regularization):** imposes a penalty term on the loss function proportional to the L2 norm of the weight vector. This results in a weight vector with small but non-zero values, and the effect is to distribute the influence of each feature across all weights. This reduces the variance of the model and prevents overfitting.

2.3.3 Activation Functions

Activation functions play a crucial role in artificial NNs. These functions facilitate the learning process by enabling the network to understand and interpret complex relationships between inputs and outputs. To enhance this flow of information, the input to the outer layer undergoes a nonlinearity process, allowing for further processing. Activation functions are vital components in this process, as they enable the network to comprehend and analyze non-linear and intricate mappings between inputs and corresponding outputs [30, 37].

- **Logistic Sigmoid:** is a widely used mathematical function employed in statistics and machine learning. It maps real-valued inputs to a bounded range between 0 and 1, making it useful for probabilistic interpretation and modeling non-linear relationships [38]. The sigmoid function is commonly used in binary classification tasks, where it produces probability estimates and facilitates the interpretation of model outputs [30]. Despite its limitations, the sigmoid function's versatility and interpretability have contributed to its widespread application, while ongoing research explores improved models and architectures [39].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

- **Hyperbolic Tangent (tanh):** maps the input values to a range between -1 and 1, providing a smooth non-linear transformation. Similar to the sigmoid function, TanH introduces non-linearity into the network, enabling it to model complex relationships between inputs and outputs. This activation function is symmetric around the origin, with negative inputs resulting in negative outputs and positive inputs producing positive outputs. The tanh

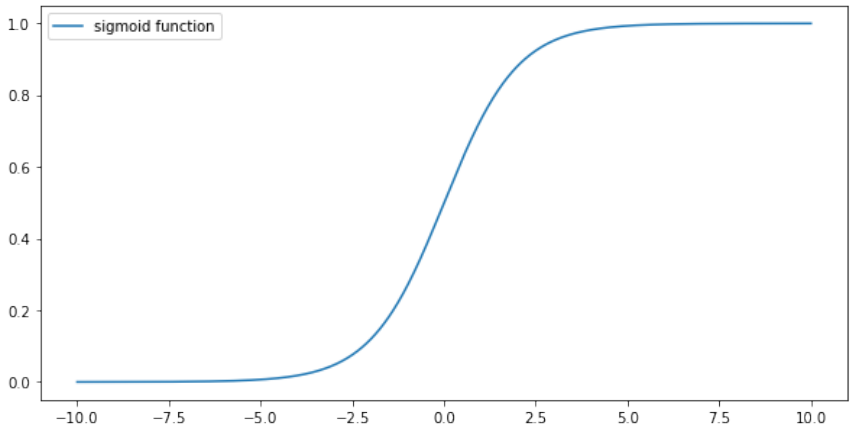

Figure 2.6: Plot of the sigmoid function

function is particularly useful in scenarios where inputs with opposing polarities need to be differentiated, making it suitable for tasks such as classification problems.

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.6}$$

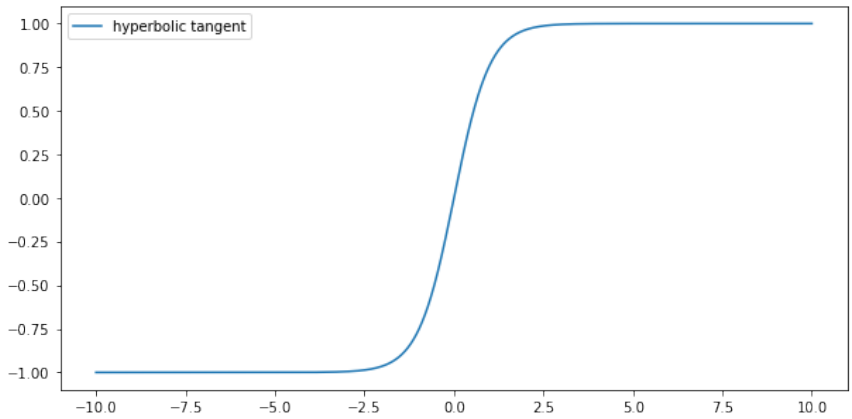

Figure 2.7: Plot of the tanh function

- **Rectified Linear Unit (ReLU):** also known as rectifier or ramp function, it is simply the positive part of its argument.

$$f(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Kunihiko Fukushima introduced this activation function in 1969, specifically within the framework of hierarchical neural networks aimed at extracting visual features [40]. As of 2017, the rectifier activation function emerged as the preferred choice for deep neural networks, gaining significant popularity [41].

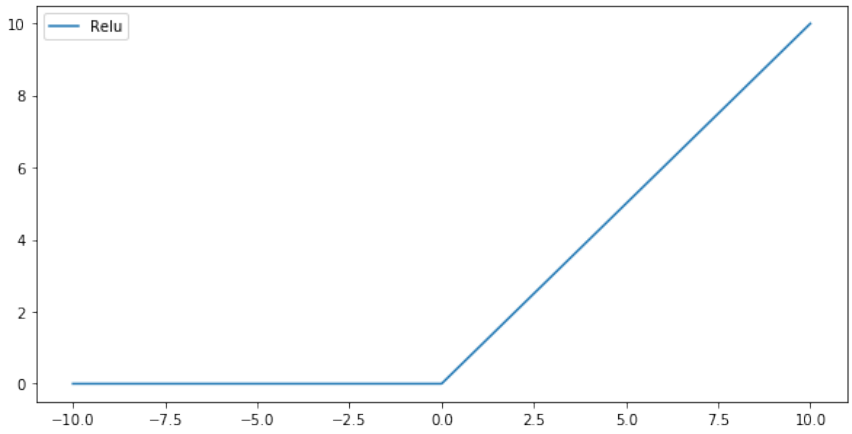

Figure 2.8: Plot of the ReLU function

- **Softmax:** transforms a vector of real numbers into a probability distribution over multiple classes. The softmax function normalizes the input values, exponentiates them, and divides each element by the sum of all exponentiated values. This ensures that the resulting probabilities are positive and sum up to one, enabling the interpretation of the output as class probabilities. The softmax function is often used in multiclass classification tasks to obtain the most likely class assignment based on learned patterns and model predictions [30, 38]. Much like other loss functions, recent research posits different ways of improving performance of the softmax function for various classification tasks [42].

$$\sigma(\mathbf{z}_i) = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{K} e^{\mathbf{z}_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_K) \in \mathbb{R}^K \tag{2.8}$$

2.3.4 Loss Functions

The primary objective of a loss function in deep learning is to quantify the discrepancy between the predicted output of a neural network and the ground truth label associated with a given input sample [30]. By defining and optimizing an appropriate loss function, deep learning models can effectively learn from data and adapt their internal parameters to minimize the discrepancy between predicted and desired outputs. Loss functions provide a quantitative measure of the model's performance, enabling all sorts of predictive tasks [29].

- **Mean Average Error:** is a commonly used loss function in regression tasks. It measures the average absolute difference between the predicted values and the true values (Eq. 2.9). By taking the absolute difference, MAE provides a robust measure of the overall prediction accuracy, unaffected by outliers [30]. MAE is particularly useful when the magnitude of errors is crucial, as it directly reflects the average deviation between predictions and ground truth values.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.9}$$

- **Mean Squared Error:** is another widely employed loss function in regression tasks and sometimes classification tasks [43]. It computes the average squared difference between the predicted values and the true values (Eq. 2.10). Squaring the differences amplifies the impact of larger errors, making MSE more sensitive to outliers compared to MAE. By penalizing larger errors more heavily, MSE places greater emphasis on minimizing significant deviations between predictions and ground truth values [30].

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.10}$$

- **Cross-entropy:** is a versatile loss function that is predominantly utilized in classification tasks, which is our case. It quantifies the dissimilarity between predicted class probabilities and the true class labels (Eq. 2.11). Cross-entropy is based on information theory and is particularly effective when dealing with probabilistic predictions. In binary classification scenarios, binary cross-entropy is commonly employed, while in multi-class classification, categorical cross-entropy is often used [30]. By maximizing the likelihood of correct class predictions, cross-entropy encourages the model to assign higher probabilities to the correct classes, facilitating accurate classification [29, 30].

$$\text{CrossEntropy} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{2.11}$$

2.3.5 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of artificial neural network that is specifically designed to process sequential data. They are widely used in various fields, including natural language processing, speech recognition, machine translation, and time series analysis. The main motivation behind RNNs is their ability to capture and utilize contextual information from previous steps or inputs in a sequence [30]. Unlike traditional feedforward neural networks, RNNs have loops in their architecture, allowing them to maintain an internal memory that retains information about past inputs. This memory enables RNNs to effectively model and process sequences of arbitrary length [30, 38].

In Fethi M. Salem's book [38], he puts it differently by stating that RNNs can be seen as a type of finite-time horizon mappings that unfold into layered feedforward networks with shared parameters across the layers (see figure 2.9). They are primarily used for mapping tasks and differ from feedback neural networks, such as Hopfield networks[16], which are used for storing and retrieving associative memories. Feedback neural networks, in contrast, are RNNs that operate over an infinite-time horizon and typically do not require external input [38].

[16]Hopfield networks are a type of recurrent neural network named after its inventor, John Hopfield. They were introduced in the 1980s and are primarily used for associative memory tasks like pattern recognition and content addressable memory [44].

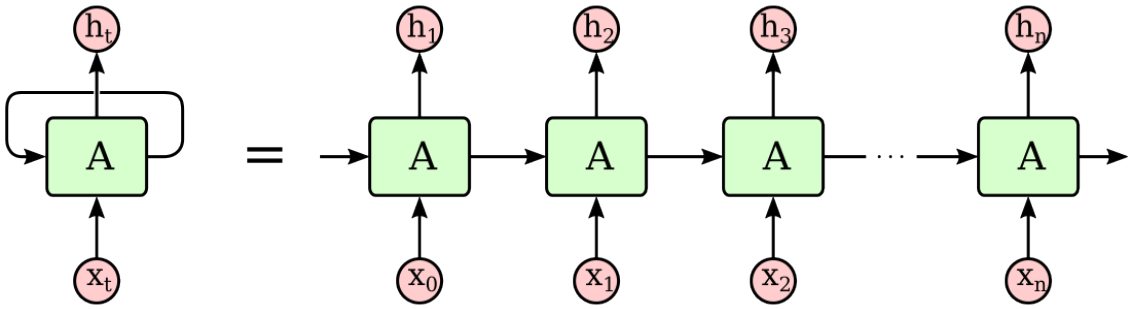

Figure 2.9: Unrolling an RNN through time / Jakub Kvita, kvitajakub.github.io

The computation within an RNN can be described by the following equations:

$$h(t) = f(\mathbf{W}x^{(t)} + \mathbf{U}h^{(t-1)} + b) \tag{2.12}$$

$$y(t) = g(\mathbf{V}h^{(t)} + c) \tag{2.13}$$

where $x(t)$ is the input vector at time step t, $h(t)$ is the hidden state vector at time step t, $y(t)$ is the output vector at time step t, f is the activation function for the hidden state, g is the activation function for the output, $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{V}$ are weight matrices to be learned, b and c are bias vectors. These equations define the recurrent nature of the network, as the hidden state at each time step is influenced by the previous hidden state. The output at each time step depends on the current hidden state [38].

2.3.5.1 Challenges

Vanilla (as in traditional) recurrent neural networks face several challenges that can impact their performance and training process. Some of the main problems encountered in RNN research are:

- **Vanishing and Exploding Gradients:** RNNs can suffer from the vanishing or exploding gradient problem during backpropagation. As information propagates through time steps, gradients can become extremely small or large, making it challenging for the network to effectively learn and capture long-term dependencies [38, 45].
  - Vanilla RNNs struggle to capture long-term dependencies in sequences. The influence of earlier inputs on the current hidden state diminishes over time due to repeated multiplication of weight matrices, making it challenging to retain information over long sequences [45].
- **Lack of Memory:** vanilla RNNs have limited memory capacity as they rely solely on their hidden state to retain information about previous inputs. This limited memory makes it difficult for the network to remember relevant information from distant time steps [38].
- **Inability to Handle Varying Sequence Lengths:** vanilla RNNs require fixed-length input sequences. If the input sequences have varying lengths, they either need to be truncated

or padded with zeros, leading to inefficient representation and potential loss of important information [30].

- **Lack of Explicit Context Separation:** vanilla RNNs do not explicitly distinguish between short-term and long-term dependencies [46]. They treat all past inputs equally, which can limit their ability to differentiate between relevant and irrelevant context [38].

To address these challenges, various advanced RNN architectures have been developed, such as Long Short-Term Memory (LSTM) networks [23] and Gated Recurrent Units (GRUs) [24]. These architectures incorporate mechanisms to mitigate the vanishing gradient problem, handle long-term dependencies, and maintain better memory.

2.3.5.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) is an advanced type of RNN architecture that addresses the vanishing gradient problem and enables better modeling of long-term dependencies in sequential data. LSTMs achieve this by incorporating memory cells and gating mechanisms that regulate the flow of information [23, 38].

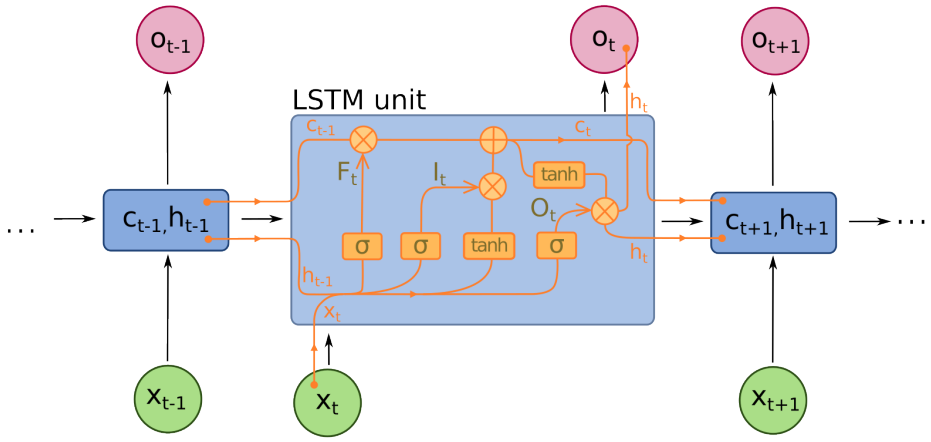

Figure 2.10: The architecture of an LSTM cell, fdeloche / CC-BY-SA-3.0

LSTMs are composed of memory cells C_t, input gates i_t, forget gates f_t, and output gates o_t. The memory cells serve as a long-term memory component, while the gates control the flow of information into, out of, and within the memory cells [23]:

$$f_t = \sigma(\mathbf{W_f} x_t + \mathbf{U_f} h_{t-1} + \mathbf{b_f}) \tag{2.14}$$

$$i_t = \sigma(\mathbf{W_i} x_t + \mathbf{U_i} h_{t-1} + \mathbf{b_i}) \tag{2.15}$$

$$o_t = \sigma(\mathbf{W_o} x_t + \mathbf{U_o} h_{t-1} + \mathbf{b_o}) \tag{2.16}$$

$$\tilde{C}_t = \tanh(\mathbf{W_c} x_t + \mathbf{U_c} h_{t-1} + \mathbf{b_c}) \tag{2.17}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{2.18}$$

$$h_t = o_t \odot \tanh(C_t) \tag{2.19}$$

2.3.5.3 Gated Recurrent Units

Gated Recurrent Units (GRUs) are another type of advanced RNN architecture that, like LSTMs, address the vanishing gradient problem and capture long-term dependencies [24, 38].

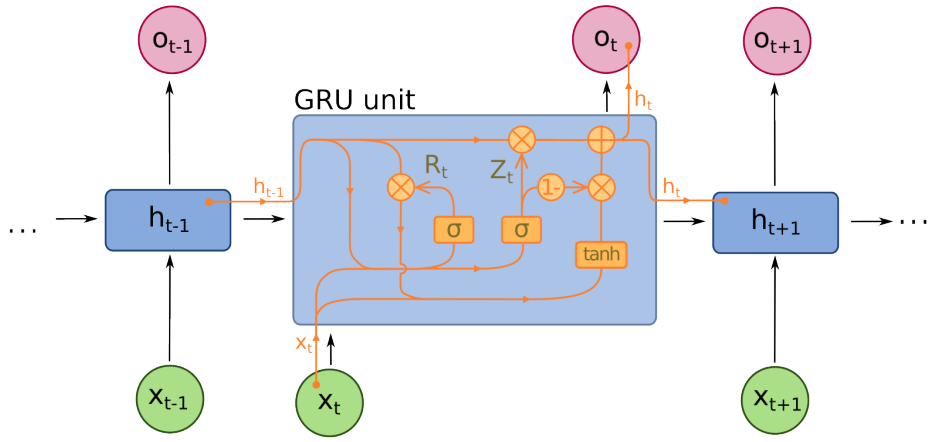

Figure 2.11: The architecture of a GRU cell, fdeloche / CC-BY-SA-3.0

GRUs simplify the architecture compared to LSTMs by combining the memory cell and the hidden state into a single unit, resulting in a more computationally efficient model [24]:

$$z_t = \sigma(\mathbf{W_z} x_t + \mathbf{U_z} h_{t-1} + \mathbf{b_z}) \tag{2.20}$$

$$r_t = \sigma(\mathbf{W_r} x_t + \mathbf{U_r} h_{t-1} + \mathbf{b_r}) \tag{2.21}$$

$$\tilde{h}_t = \tanh(\mathbf{W_h} x_t + \mathbf{U_h}(r_t \odot h_{t-1}) + \mathbf{b_h}) \tag{2.22}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.23}$$

2.3.6 Attention Mechanism

Attention is a mechanism that allows a neural network to focus on relevant parts of the input or output while ignoring the irrelevant ones. It is inspired by the human cognitive process of selectively concentrating on a few things while ignoring others [47, 48]. Attention is important because it can improve the performance and interpretability of neural networks, especially in natural language processing (NLP) and computer vision tasks. It can be used when the input or output sequences are long, complex, or have variable lengths, which is usually the case for machine translation, text summarization, or image captioning [47].

2.3.6.1 Soft vs. Hard Attention

- **Hard attention:** selects one input state for each output state, and uses it as the context vector. This way, it can focus on only one input state at a time. Hard attention is non-differentiable and requires reinforcement learning or variational inference to train [47].
- **Soft attention:** assigns a probability distribution over all the input states for each output

state, and takes a weighted average of them as the context vector. This way, it can attend to all the input states to some degree. Soft attention is differentiable and can be trained end-to-end with backpropagation [48]. As we only care about supervised learning for the sake of this thesis, we will only present how to achieve the soft-attention mechanism:

1. First, a differentiable function a, such as a dot product, a feed-forward network, or a bilinear form, is applied to compute the alignment scores between states. s_{t-1} is the initial context vector.
$$e_{t,i} = a(s_{t-1}, h_i) \tag{2.24}$$

2. Then, a softmax or a sigmoid function is used to normalize the scores into a probability distribution. This enables the backpropagation of the gradients through the attention mechanism and the end-to-end training of the model.
   - Using the softmax function:
$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})} \tag{2.25}$$
   - Using the sigmoid function parametered with $\mathbf{q} \in \mathbb{R}^d$:
$$\alpha_{t,i} = \mathbf{q}^\top \sigma(e_{t,i}) \tag{2.26}$$

3. Finally, a weighted average of the input states using the probability distribution as the weights.
$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i \tag{2.27}$$

where $\alpha_{t,i}$ is the alignment score between the decoder state s_{t-1} and the encoder state h_i, $e_{t,i}$ is the unnormalized score computed by a function a, and c_t is the context vector for time step t.

2.3.6.2 Global vs. Local Attention

- **Global attention:** computes the alignment between all the encoder states and the current decoder state, it can capture long-range dependencies, but it is computationally expensive.
- **Local attention:** only focuses on a subset of the encoder states. It can reduce the computation cost, but it may miss some relevant information.

2.4 Graph Neural Networks

A graph neural network (GNN) is a type of neural network that operates on graph data. It can learn from the features and structure of the graph to perform various tasks. consisting of three main components: an input layer, one or more GNN layers, and an output layer.

GNNs are useful for analyzing complex and interrelated data that cannot be easily handled by traditional neural networks due to the permutation invariance problem. They can leverage both the features and structure of graphs to learn rich and expressive representations that capture the patterns and dependencies in the data. They can also generalize well to unseen or new graphs by learning from local neighborhoods rather than global structures.

A straightforward way to construct a model that processes graphs is to take the adjacency matrix as input to a deep neural network. However, this method suffers from the drawback that it relies on the arbitrary ordering of nodes in the adjacency matrix. In other words, such a model lacks *permutation invariance* or *equivariance*, which are essential properties for designing neural networks over graphs. Mathematically, any function that operates on an adjacency matrix should ideally satisfy one of the two following properties [49]:

- *Permutation invariance:* implies that the function is independent of the arbitrary ordering of the rows/columns in the adjacency matrix.
- *Permutation equivariance:* implies that the output of f is permuted in a consistent way when we permute the adjacency matrix.

Achieving invariance or equivariance is a key challenge when we are learning over graphs [49].

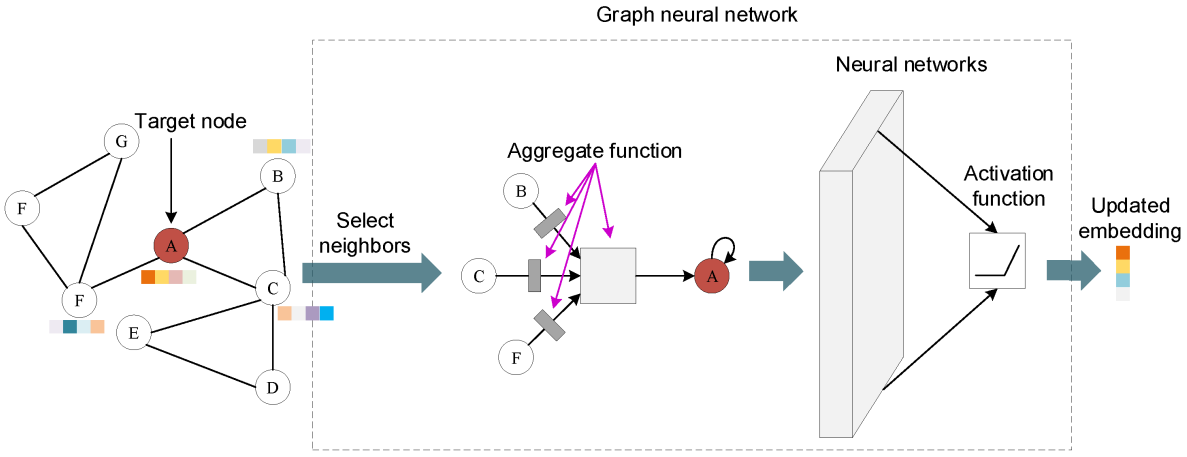

Figure 2.12: Basic GNN process from input to output

2.4.1 Graph Theory Basics

Graph theory is a branch of mathematics that studies the properties and applications of graphs, which are structures that consist of a set of vertices and a set of edges. Vertices are the basic units

of a graph, and edges are the connections between them. Graphs can be used to model many real-world phenomena: social networks, maps, molecules, etc. Some useful properties of graphs are as follows:

- **Degree:** the number of edges incident to a vertex. A vertex with degree zero is called isolated.
- **Path:** a sequence of vertices connected by edges. A path is simple if it does not repeat any vertices, except possibly the first and the last one.
- **Cycle:** a path that starts and ends at the same vertex. A graph is acyclic if it has no cycles.
- **Directed graph:** a graph where each edge has a direction, indicating the order of the vertices. A directed graph is acyclic if it has no directed cycles.
- **Undirected graph:** a graph where each edge has no direction, meaning that the order of the vertices does not matter. An undirected graph is connected if there is a path between any pair of vertices.

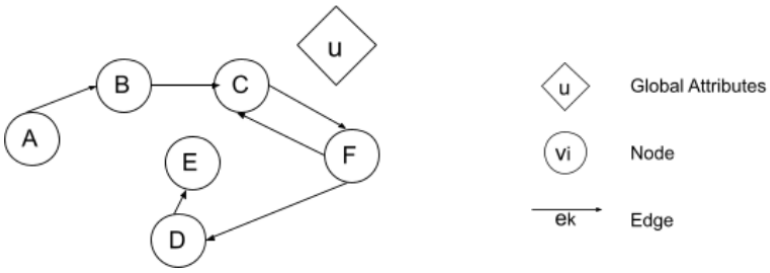

Figure 2.13: Simple graph notation

2.4.2 Applications

GNNs have a wide range of applications in various fields and tasks. Depending on the type of GNN, they can address different problems: node classification, graph classification, graph generation, network embedding, and spatial-temporal graph forecasting [4]. We present some of the applications based on the following research areas:

- **Social networks:** GNNs have been used to model social networks and analyze social relationships. For instance, they have been used to predict the next user action on social media platforms, recommend friends, detect fake news, and identify communities [50].
- **Molecular structure:** GNNs are used to model molecular structures and predict molecular properties. They have applications in drug discovery, material science, and chemistry [51].
- **Computer vision:** GNNs have been applied to computer vision tasks such as object detection, semantic segmentation, and image classification [52]. GNNs can also be used for visual tracking and object recognition in videos.

- **Natural language processing:** GNNs can be used for text classification, named entity recognition, and sentiment analysis. They can also be used to build chatbots and question-answering systems [53].
- **Recommendation systems:** GNNs can be used to develop personalized recommendation systems that can recommend items to users based on their behavior and preferences [3, 5].

GNNs have shown promising results in various domains, and their applications are still expanding with the development of new GNN architectures and algorithms.

2.4.3 Tasks

- **Node-level classification:** assigns labels or values to individual nodes in a graph. This can be achieved by GNNs that learn node embeddings through information diffusion or graph convolution like RecGNNs (see Sub-subsection 2.4.5.1) and ConvGNNs (see Sub-subsection 2.4.5.2). By adding a fully connected layer or a softmax layer at the end, GNNs can perform node-level classification in a holistic fashion [4].
- **Edge-level classification:** infers the label or the weight of a link between two nodes in a graph. This can be done by using a similarity measure or a neural network that takes the node embeddings from GNNs as inputs and outputs the edge properties [4].
- **Graph-level classification:** is to assign a label to an entire graph. To achieve this, GNNs often employ pooling and readout mechanisms (see Subsubsection 2.4.4.2) to compress the graph information into a single vector [4].

2.4.4 Message Passing Framework

The term graph neural networks encompasses any neural network that operates on graphs and produces representations of nodes or graphs. Due to the rapid advances in this field, various GNN architectures have been proposed with different mechanisms for learning from graph-structured data. However, these architectures differ in their methods of updating and aggregating node features, which makes them challenging to understand and compare[17]. To address this issue, a framework was developed to unify and standarize them under three common functions. This framework is known as the Message-Passing Framework [51].

Message passing neural networks (MPNNs) provide a framework that facilitates the iterative aggregation of information from neighboring nodes, enabling the transmission of messages not only within the local neighborhood but also to other neighborhoods that are not directly connected. The MPNN framework can encompass at least eight prominent models from the literature [51]. For the sake of our thesis, we will focus on MPNNs that deal with undirected graphs with node features and edge features, but the formalism can be easily generalized to directed multigraphs.

[17]Note that some GNNs, known as *spatial*, are not based on message passing.

The forward pass consists of two phases: *a message passing phase* and *an optional readout phase* [54].

2.4.4.1 Message Passing Phase

The message passing phase lasts for T iterations and involves message functions M_t and vertex update functions U_t. In this phase, hidden states h_v^t of each node in the graph are updated based on messages m_v^{t+1} as follows:

1. **The message function:** is responsible for aggregating information from the neighboring nodes of a given node in a graph, and producing a message vector that captures the local context of that node [54].

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2.28}$$

   This process can alternatively be conceptualized as the composition of two functions: message formulation and aggregation. However, the objective is still consistent across different formulations.

2. **Update function:** is responsible, on the other hand, for updating the state of a given node in a graph, based on the message vector produced by the message function m_v^{t+1} and the previous state of the node [54]. The update function can be a simple linear transformation or a more complex recurrent neural network that learns how to integrate the message vector and the previous state (see Subsection 2.4.5.1). The aim of the update function is to learn new node representations that capture the global context of the node after receiving messages from its neighborhood [51, 55].

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{2.29}$$

The MP network consists of different layers that implement each step. The network depth corresponds to the number of iterations. The final feature vector h_v^T of v incorporates messages from all the nodes in the sub-tree of height T that has v as its root. It reflects both the structure of the neighborhood of v and the distribution of the vertex representations within it [55].

2.4.4.2 Readout Phase

A permutation-invariant readout pooling function R is employed to obtain a feature vector for the whole graph, which can be used for tasks such as classification or regression [55, 56]:

$$\hat{y} = R(h_v^t | v \in G) \tag{2.30}$$

The optimal readout function depends on the dataset characteristics, which makes it challenging to select the best function for a given task and often results in suboptimal performance. Previous methods to reduce the information loss at the readout phase have employed four different types of techniques [56]:

- **Fuzzy histograms:** aggregate the vector embeddings by computing the membership probabilities to various fuzzy bins [57].
- **Hierarchical clustering:** leverage the hierarchical structures present in graphs [58].
- **Multi-Layer Perceptrons:** can also be seen as weighted sums of node features and have high expressiveness but lack order invariance [27, 56].
- **Attention Mechanisms:** node representations are fed as input to the encoder of the attention mechanism and the output of the decoder is the graph representation. The *context*[18] and type of the attention can be domain specific [59].

2.4.5 GNN Architectures

Wu *et al.*'s GNN survey [4] proposed a classification scheme for GNNs. They divided GNN architectures into four main categories: *RecGNNs*, which use recurrent neural networks to update node representations; *ConvGNNs*, which apply convolutional operations on graph signals; *GAEs*, which learn low-dimensional embeddings of graphs; and *STGNNs*, which capture spatial and temporal dependencies in dynamic graphs. For the sake of this thesis, we will limit ourselves to RecGNNs, ConvGNNs and Graph Attention Networks (GATs).

2.4.5.1 Recurrent Graph Neural Networks (RecGNNs)

Recurrent Graph Neural Networks (RecGNNs) are among the earliest works of graph neural networks [4]. RecGNNs aim to learn node representations with recurrent neural architectures. They assume a node in a graph continuously communicates with its neighbors until a stable equilibrium is achieved. RecGNNs are conceptually significant and inspired subsequent research on convolutional graph neural networks [60].

$$h_v^t = RNNCell(h_v^{(t-1)}, m_v^t) \tag{2.31}$$

RecGNNs employ an RNN cell as an update function, the message function can differ depending on the model and the graph structure. The RNN cell takes the node representations of the previous iteration h_v^{t-1} as its memory vector, and the message m_v^t as its input to compute h_v^t. GGNNs use advanced RNN cells like LSTMs (see Sub-subsection 2.3.5.2) or GRUs (see Sub-subsection 2.3.5.3) to exploit their gating mechanisms to update the node representations [27, 60].

[18] Context in attention mechanisms is a vector that represents the relevant information from the input data for a given task [48].

2.4.5.2 Graph Convolutional Networks (ConvGNNs)

ConvGNNs, or GCNs, mainly utilize the pooling operation to integrate information from node v's neighbourhood node w in the graph to help with the update of the hidden state of n_i, different specific pooling operations including mean pooling and max pooling can be utilized, depending on the specific scenarios.

$$\hat{h_v^t} = pooling(h_w^{(t-1)} | w \in N(v)) \tag{2.32}$$

Both node neighborhoods in a ConvGNN and convolutions in a CNN are methods of learning from local information, but they differ in how they define and process the local regions [4]. A CNN uses fixed-size patches that are determined by the filter size and stride, while a ConvGNN uses variable-size neighborhoods that are determined by the graph structure and hop distance. A typical CNN uses matrix multiplication and element-wise operations to apply the filter to the patches, while a ConvGNN uses various aggregation functions to combine the features of the neighbors [61].

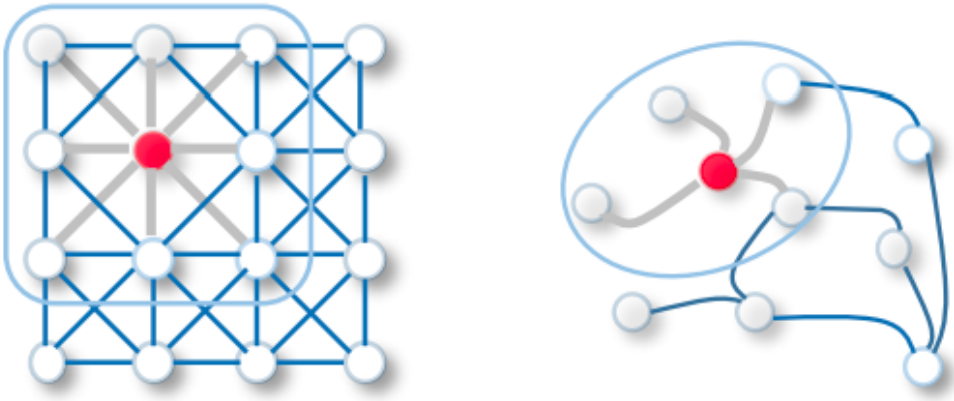

Figure 2.14: Node neighborhoods are similar to convolutions in a CNN [4]

Subsequently, the node n_v's hidden state can be updated iteratively by incorporating the aggregated information from its neighbourhood.

$$h_v^t = h_v^{(t-1)} + \hat{h}_v^t \tag{2.33}$$

2.4.5.3 Graph Attention Networks (GATs)

Graph Attention Networks (GAT) [62] utilize mechanisms to learn the relative weights between two connected nodes. The graph convolutional operation according to GAT is defined as

$$h_v^t = attention(h_w^{(t-1)}, w \in N(v)) \tag{2.34}$$

The *attention* module is a generic module that can be instantiated with various operations such

as self attention, multi-head attention, and so on. The *attention* module essentially performs two steps: (1) computing the relevance scores of each node in the neighbourhood, and (2) combining the hidden states of the nodes in the neighbourhood based on their relevance scores [3, 4].

2.5 Gated Graph Neural Networks

Gated Graph Neural Networks are a pioneering RecGNN architecture [27] that employ a GRU cell as the update function. GGNNs implement the message passing mechanism for graph neural networks and update the node representations based on the adjacency matrices and the linear transformations applied on them.

Crucially, the parameters of this RNN-style update are always shared across nodes (i.e., the same LSTM or GRU cell are used to update each node). In practice, researchers usually share the parameters of the update function across the message-passing layers of the GNN as well. These gated updates are very effective at enabling deep GNN architectures (e.g., more than 10 layers) and avoiding over-smoothing [27, 49]. Generally, they are most useful for GNN applications where the prediction task requires complex reasoning over the global structure of the graph, such as applications for program analysis [27] or combinatorial optimization [63].

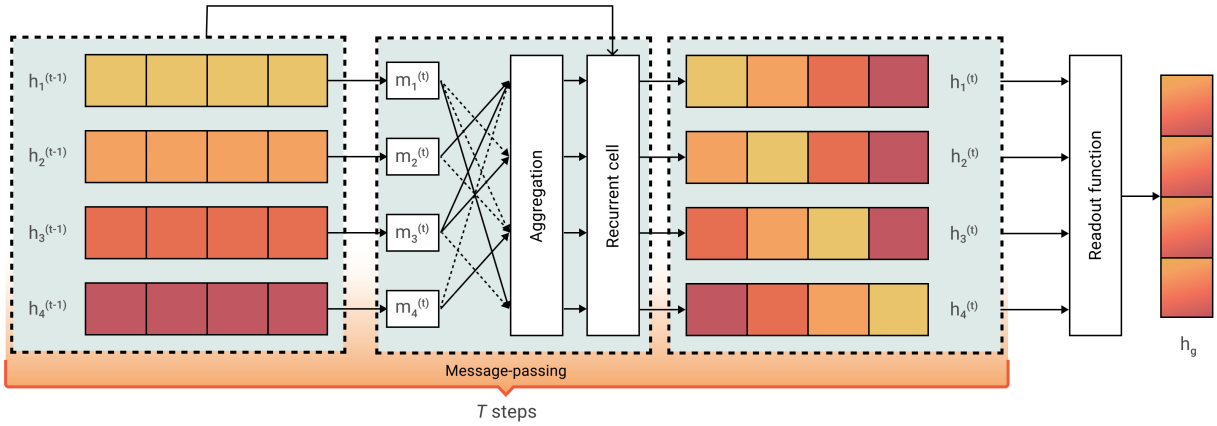

Figure 2.15: The architecture of a GGNN layer with a readout function

2.5.1 Node Annotations

In GNNs, the node representations do not need to be initialized because the fixed point is independent of the initializations due to the contraction map constraint. This is not true for GGNNs, which allow us to incorporate node labels as additional inputs [27]. This essentially means that we can use domain-specific node information as node embeddings to enhance their representations.

For each node v in the graph, $A_{v:} \in \mathbb{R}^{D|V| \times 2D}$ as the pair of column blocks that correspond to the incoming and outgoing adjacency matrices (see figure 2.16) are defined.

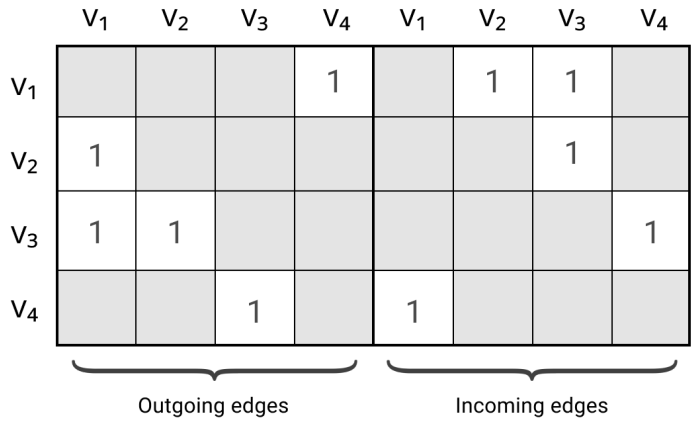

Figure 2.16: The formulation of the A matrix, which is the concatenation of the ougoing and incoming edges

2.5.2 Propagation Model

$$h_v^1 = [x_v^\top, 0, \ldots, 0]^\top \tag{2.35}$$

$$a_v^t = A_{v:}^\top [h_1^{(t-1)\top}, \ldots, h_{|V|}^{(t-1)\top}]^\top + \mathbf{b} \tag{2.36}$$

$$h_v^t = GRU(h_v^{(t-1)}, a_v^t) \tag{2.37}$$

The first equation (Eq. 2.35) initializes the hidden state of each node by copying its annotation into the first component and filling the rest with zeros. The second equation (Eq. 2.36) propagates information across the graph through the incoming and outgoing edges, using edge type and direction specific parameters. The vector $a_v \in \mathbb{R}^{2D}$ aggregates the activations from both directions of the edges [49]. The final equations (Eq. 2.37) update the hidden state of each node by integrating information from the other nodes and from the previous timestep, using a GRU-like mechanism [27].

2.6 Limitations

Graph-based Neural Networks (GGNNs) have gained attention as powerful models for learning representations and making predictions on graph-structured data. However, they also come with certain limitations. Here are some of the key limitations of GGNNs:

- **Computational Complexity:** GGNNs typically operate on the entire graph structure simultaneously, which can lead to high computational complexity. The number of iterations required to update node representations in GGNNs is proportional to the diameter of the

graph, making them less efficient for large graphs with long paths [27].

- **Limited Memory Capacity:** GGNNs have limited memory capacity, as they typically employ fixed-size hidden state vectors to represent nodes in the graph. This can restrict their ability to capture and retain information from large or complex graphs.
- **Sensitivity to Graph Structure:** GGNNs are sensitive to the input graph structure, meaning that small changes in the graph can have a significant impact on the learned representations and predictions. This sensitivity can limit their generalization performance when applied to graphs with different structures or when dealing with noisy or incomplete graphs [4].
- **Lack of Explicit Handling of Graph Properties:** GGNNs primarily focus on learning node representations and capturing local graph structure information. They may struggle to capture global properties or higher-order interactions in the graph, such as community structures or long-range dependencies [4].
- **Interpretability Challenges:** GGNNs often lack interpretability due to their complex and black-box nature. Understanding the learned representations and the reasoning behind model predictions can be challenging, which may limit their applicability in domains where interpretability is crucial [30].

2.7 Conclusion

In this chapter, we have explored the fascinating world of GNNs, a powerful technique for analyzing and inferring properties from graphs. We have seen how GNNs fit within the broader context of artificial intelligence and graph analysis, and how they leverage gated mechanisms to propagate messages between nodes and capture both local and global dependencies. We have also discussed the principles and applications of attention mechanisms, which enhance the network's ability to extract meaningful insights from graphs. By reimagining existing reading systems from a blind perspective, GNNs open up new possibilities and challenges for graph-based learning and reasoning. One such possibility is the use of GGNNs coupled with the soft-attention mechanism for recommendation tasks, especially for Session-Based Recommendation, which we will explore in the next chapter.

CHAPTER THREE:

Session-Based Recommender Systems using Gated Graph Neural Networks

3 Session-Based Recommender Systems using Gated Graph Neural Networks

3.1 Introduction

Unlike traditional recommendation methods that rely on user profiles and long-term preferences, session-based recommendation has to deal with anonymous users and dynamic behavior. To address this challenge, session data can be mapped to graphs. We use GNNs to learn node representations of the items in a session graph, which can encode both the sequential and collaborative information of user behavior. Moreover, we use a soft-attention mechanism as a readout function to aggregate the node representations into a session representation. The soft-attention mechanism can capture both the global and local information of the session graph by assigning different weights to different nodes according to their relevance to the last item in the session. We then use the session representation to compute the scores of candidate items and recommend the top-N items with the highest scores. We evaluate this method on two real-world datasets and show that it outperforms several state-of-the-art baselines for session-based recommendation.

3.2 Session-Based Recommender System using Gated Graph Neural Networks

3.2.1 Session Graph Connection Scheme

Session graphs are constructed by modeling pairwise item-transitions within the current session s. Each node $v_{s,i}$ in the session graph $G_s = (V_s, E_s)$ represents an item, and each edge $(v_{s,i-1}, v_{s,i}) \in E_s$ represents a transition between two items.

We map each item $v \in V$ to a latent vector $\mathbf{v} \in \mathbb{R}^d$ in an unified embedding space, where d is the dimensionality. This mapping will be learned by GNNs later. Then, we obtain an embedding vector $\mathbf{s}$ for each session s by aggregating the latent vectors of the items in that graph.

3.2.1.1 Edge Occurrence over Source Outdegree Ratio Normalization

In the SR-GNN model [5], the session graph is built from the session sequence by first laying each unique item in the sequence as a node, and then connecting them with a directed edge $(v_{s,i-1}, v_{s,i})$ if $v_{s,i-1}$ occurs directly before $v_{s,i}$ in the session sequence. The resulting is a multigraph[19], as a transition between two items can occur multiple times in the session. Finally, a normalized weight is appointed to each edge $(v_{s,i-1}, v_{s,i})$, which is calculated as its occurrence in the session divided by the outdegree of its source node $deg^+(v_{s,i})$. As such, for the sequence $s_A = \{v_1, v_2, v_3, v_2, v_4, v_2, v_5\}$,

[19] A multigraph is a graph that allows for multiple parallel edges, i.e., two vertices can be connected by more than one edge in the same direction.

the session graph is formulated as can be seen in figure 3.1 along with its adjacency matrix in figure 3.1.

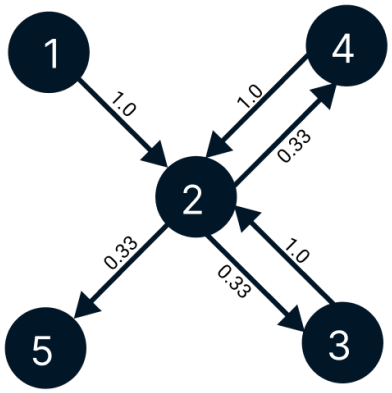

Figure 3.1: Resulting graph from s_A

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1/3	1/3	1/3
3	0	1	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

Figure 3.2: Adjacency matrix of s_A

3.2.1.2 Relevant Order Graph Formulation

Chen et al. [64] identified two issues regarding the representation of session graphs in the literature, the *Lossy Session Encoding* being one of them. Essentially, the session encoding method we previously described introduces information loss in the form of sequence order being unsuccessfully mapped to the session graph, i.e., given a session graph G_s, we cannot always determine the order of the sequence that it emerges from. This implies that the session encoding is not injective[20] in its formulation of graphs.

To demonstrate this, we present a counter-example; not only does s_A result in the graph in the figure, another sequence $s_B = \{v_1, v_2, \underline{v_4}, v_2, \underline{v_3}, v_2, v_5\}$ also produces it. Note that if we were to permutate the underlined items, we get s_A.

To remedy this, for every node $v \in V_s$, order each incoming edge $E_s^{(in)}(v)$ by the order of its occurrence in the session. We keep track of the order by assigning a counter[21] attribute to each edge in $E_s^{(in)}(v)$ that indicates its order with respect to the edges in $E_s^{(in)}(v)$. We illustrate this graph formulation method in figure 3.3, along with its Python implementation (Listing 1).

[20] Any function that associates at most one preimage element with each image element.
[21] Casted to float in practice.

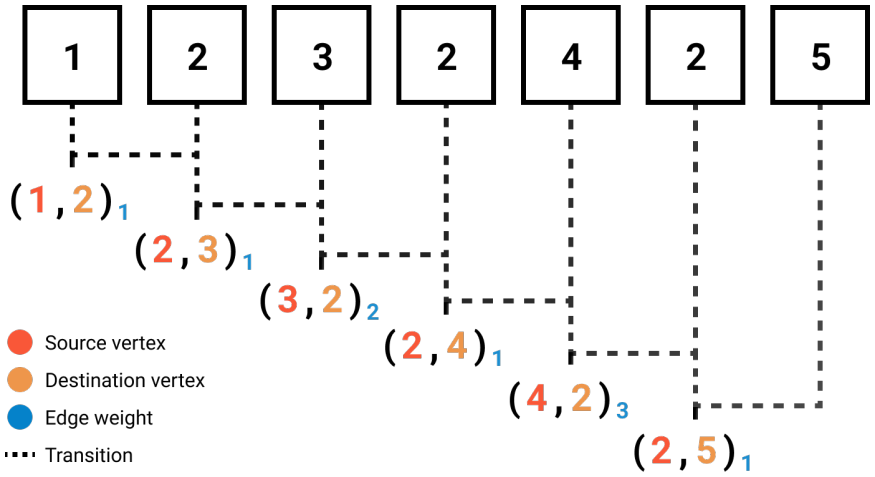

Figure 3.3: Graph formulation algorithm applied on session s_A

```python
def seq_to_graph(seq):
    length = len(seq)

    edges = []
    nodes = set(seq)
    order = dict.fromkeys(nodes, 0.0)

    for i in range(length - 1):
        order[seq[i + 1]] += 1.0
        edges.append((seq[i], seq[i + 1], order[seq[i + 1]]))

    return (nodes, edges)
```

Listing 1: Python implementation of the graph formulation algorithm figure 3.3

By doing so, we add sequence order relevance to our graph. Our previous session sequences now result in different graphs as shown in figure 3.4 and figure 3.5.

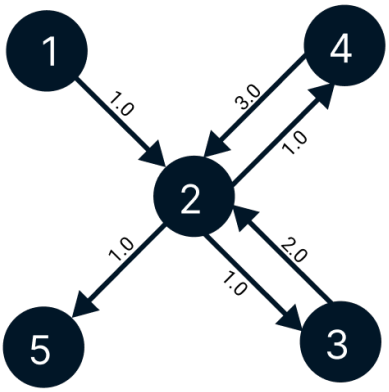

Figure 3.4: Lossless session graph of s_A

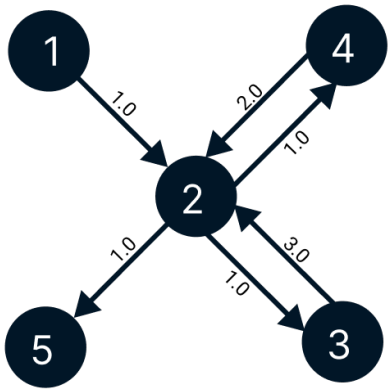

Figure 3.5: Lossless session graph of s_B

This alone does not fix the problem entirely. For example, let's suppose we have two cyclic session sequences $s_{A'} = \{v_1, v_2, v_3, v_1\}$ and $s_{B'} = \{v_2, v_3, v_1, v_2\}$, their cyclic nature results in them having the same vertices and edges regardless of the order. To render the encoding truly lossless, the last item has to be distinguished in the architecture of the model, and subsidiarily indicate its relevance to the prediction.

Finally, normalizing the weights e_w is as trivial as dividing each weight by the max edge weight in the session graph. We will end up with $\forall e \in E_s, \;\; e_w \in [0, 1]$.

3.2.2 Architecture

Our model consists of three main layers: *a GNN layer*, *an attention layer*, and *a fusion layer* (see figure 3.6). The GNN layer takes the session graph as input and computes the node representations of the items in the session using a GRU cell. The attention layer uses a soft-attention mechanism to aggregate the node representations into a session representation, where the representation of the last item in the session serves as the query vector. The fusion layer combines the session representation and the representation of the last item into a local session representation, which captures both the global and local information of the session. Finally, we observe that the session encoding is crucial and should be distinctive to achieve better results.

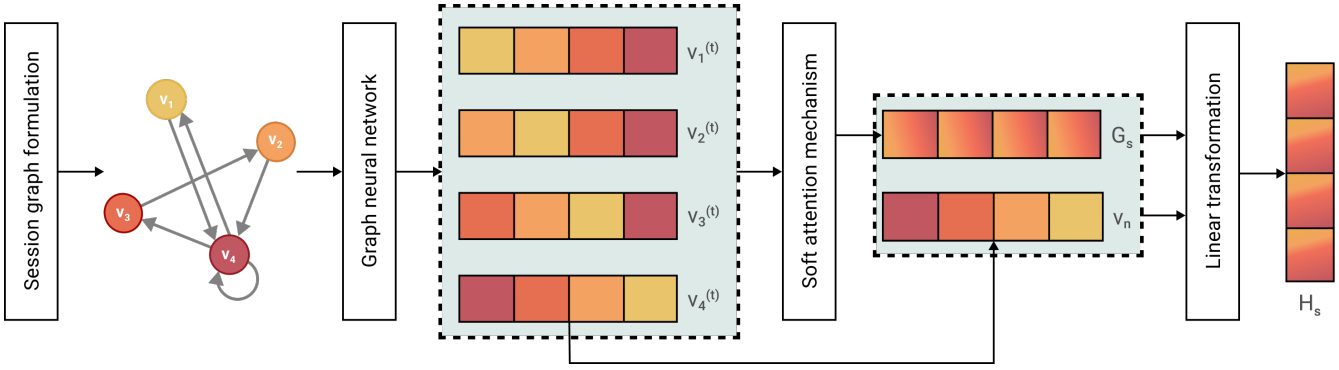

Figure 3.6: The architecture of our model

We note that the formalization of the operations in the equations and the implementation in the code differ in the shapes of the matrices and tensors. This is because we need to handle batches of sessions in the code, while the equations only deal with single sessions. However, the operations are essentially the same, and we use appropriate reshaping and broadcasting techniques to ensure that they are performed correctly and efficiently. In practice, we pad all the dimensions to match the maximum sizes of its axes. This can introduce noise in the values that we need to aggregate, but we can easily filter out the noise by multiplying our tensors with a mask before doing the aggregation computations.

3.2.2.1 Gated Graph Neural Network

$$\mathbf{a}_{s,i}^{(in),t} = \Big[\mathbf{W}_{in}[\mathbf{v}_{s,1}^{(t-1)}, \mathbf{v}_{s,2}^{(t-1)}, \ldots, \mathbf{v}_{s,n}^{(t-1)}] + \mathbf{b}_{in}\Big](\mathbf{A}_{s,i:}^{(in)})^{\top} \tag{3.1}$$

$$\mathbf{a}_{s,i}^{(out),t} = \Big[\mathbf{W}_{out}[\mathbf{v}_{s,1}^{(t-1)}, \mathbf{v}_{s,2}^{(t-1)}, \ldots, \mathbf{v}_{s,n}^{(t-1)}] + \mathbf{b}_{out}\Big](\mathbf{A}_{s,i:}^{(out)})^{\top} \tag{3.2}$$

$$\mathbf{a}_{s,i}^{t} = [\mathbf{a}_{s,i}^{(in),t}; \mathbf{a}_{s,i}^{(out),t}] \tag{3.3}$$

In listing 2, the GGNN layer performs multiple propagation steps to update the node representations of the session graph. In each propagation step, the current node representations `node_vectors` are fed into two linear neural networks $\texttt{linear\_message\_passing}_{\texttt{in}}$ and $\texttt{linear\_message\_passing}_{\texttt{out}}$, which produce two tensors of shape $\texttt{batch\_size} \times \texttt{max\_number\_of\_nodes} \times d$. These tensors are then multiplied by the corresponding adjacency matrices $\mathbf{A}^{(in)}$ and $\mathbf{A}^{(out)}$, which encode the in- and out-edges of the session graph, respectively (Eq. 3.1, 3.2). The resulting tensors $\mathbf{a}_s^{(in),t}$ and $\mathbf{a}_s^{(out),t}$ are concatenated along the hidden dimension axis (Eq. 3.3).

```
# (batch_size * max_number_of_nodes, hidden_size)
node_vectors_in = self.linear_message_passing_in(_node_vectors)
node_vectors_out = self.linear_message_passing_out(_node_vectors)

# (batch_size, max_number_of_nodes, hidden_size)
a_in = tf.matmul(adj_in, node_vectors_in)
a_out = tf.matmul(adj_out, node_vectors_out)

# (batch_size, max_number_of_nodes, hidden_size * 2)
a = tf.concat([a_in, a_out], axis=-1)
```

Listing 2: GGNN message phase

Then, in listing 3, $\mathbf{a}_s^t$ is then passed to a GRU cell with the previous node vector $\mathbf{v}_s^{(t-1)}$ as memory, which outputs a new tensor representing the updated node representations. The GRU cell allows the model to capture the temporal dynamics of the session graph and learn complex nonlinear transformations of the node features as per the GGNN spec (Eq. 3.4...3.7) [27].

$$\mathbf{r}_{s,i}^{t} = \sigma(\mathbf{W}_r \mathbf{a}_{s,i}^{t} + \mathbf{U}_r \mathbf{v}_{s,i}^{(t-1)} + \mathbf{b}_r) \tag{3.4}$$

$$\mathbf{z}_{s,i}^{t} = \sigma(\mathbf{W}_z \mathbf{a}_{s,i}^{t} + \mathbf{U}_z \mathbf{v}_{s,i}^{(t-1)} + \mathbf{b}_z) \tag{3.5}$$

$$\tilde{\mathbf{v}}_{s,i}^{t} = \tanh(\mathbf{W}_h \mathbf{a}_{s,i}^{t} + \mathbf{U}_h(\mathbf{r}_{s,i}^{t} \odot \mathbf{v}_{s,i}^{(t-1)}) + \mathbf{b}_h) \tag{3.6}$$

$$\mathbf{v}_{s,i}^{t} = (1 - \mathbf{z}_{s,i}^{t}) \odot \mathbf{v}_{s,i}^{(t-1)} + \mathbf{z}_{s,i}^{t} \odot \tilde{\mathbf{v}}_{s,i}^{t} \tag{3.7}$$

σ is the sigmoid function, $\odot$ is the element-wise multiplication, and $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{b}$ are the weight matrices and bias vectors of the GRU cell.

```
recurrent_cell = keras.layers.GRUCell(self.hidden_size)

# (batch_size * max_number_of_nodes, hidden_size)
_, _node_vectors = tf.compat.v1.nn.dynamic_rnn(
    cell=recurrent_cell,
    inputs=tf.expand_dims(a, axis=1),
    initial_state=_node_vectors,
)
```

Listing 3: GGNN update phase using a GRU cell

3.2.2.2 Soft-Attention Mechanism as Readout

The readout layer uses a soft-attention mechanism to compute the session representation. The query vector is the representation of the last item in the session, which is passed to a linear neural network $\mathtt{nn_q}$. The input vectors are the node representations of the session graph, which are passed to another linear neural network $\mathtt{nn_v}$. The outputs of these two networks are added together and then sigmoided. The resulting tensor is then fed into another linear neural network $\mathtt{nn}_\alpha$, which calculates the attention weights of each node representation with respect to the query vector (Eq. 3.8). These weights reflect how much each node influences the last item in the session. The weighted node representations are then multiplied by their corresponding weights using broadcasting and then aggregated along the node dimension to obtain the global session representation $\mathbf{g}_s$ (Eq. 3.9).

$$\alpha_{s,i} = \mathbf{W}_\alpha \sigma(\mathbf{W}_q \mathbf{v}_n + \mathbf{W}_v \mathbf{v}_i + \mathbf{b}_v) \tag{3.8}$$

$$\mathbf{g}_s = \sum_{i=1}^{n} \alpha_{s,i} \mathbf{v}_i \tag{3.9}$$

The linear transformations are performed by three weight matrices $\mathbf{W}_\alpha$, $\mathbf{W}_q$ and $\mathbf{W}_v$, and a bias vector $\mathbf{b}_v$. The weight matrices and bias vector are learned parameters of the layer.

3.2.2.3 Representation Fusion and Score Formulation

In listing 4, the fusion layer computes the hybrid representation $\mathbf{h}_s$ of the session and then the score of each candidate item for the session. $\mathbf{h}_s$ is obtained by concatenating the global session representation $\mathbf{g}_s$ and the representation of the last item $\mathbf{v}_n$ in the session. The global session representation is the output of the attention layer, which captures the global and local information of the session. The concatenated tensor is then passed to a linear neural network $\mathtt{nn_h}$, which

produces $\mathbf{h}_s$ (Eq. 3.10).

$$\mathbf{h}_s = \mathbf{W}_h[\mathbf{v}_n; \mathbf{g}_s] \tag{3.10}$$

$$\hat{\mathbf{z}}_i = \mathbf{h}_s^\top \mathbf{v}_i \tag{3.11}$$

$$\hat{\mathbf{y}}_i = \frac{\exp(\hat{\mathbf{z}}_i)}{\sum_j \exp(\hat{\mathbf{z}}_j)} \tag{3.12}$$

The score of each candidate item $\hat{\mathbf{z}}_i$ is calculated by multiplying the hybrid session representation and the item embedding using dot product (Eq. 3.11). Then, we apply a softmax function to get the output vector of the model $\hat{\mathbf{y}}$ (Eq. 3.12). The score reflects how likely the item is to be the next one in the session. The higher the score, the more relevant the item is to the session.

```
# (batch_size, hidden_size * 2)
concatenation = tf.concat([session_local_representation,
    session_graph_representation], axis=1)

# (batch_size, hidden_size)
session_representation = self.linear_transformation(concatenation)
```

Listing 4: The fusion layer's formulation of the hybrid representation

3.2.3 Training

We use cross-entropy as our loss function to measure the performance of the model. Cross-entropy is a measure from the field of information theory that calculates the difference between two probability distributions. For our case, the two distributions are the true labels of the items and the predicted probabilities of the items by the model. The cross-entropy loss function can be defined as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i \log(\hat{\mathbf{y}}_i) + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i) \tag{3.13}$$

where N is the number of items, $\mathbf{y}_i$ is the true label of the i-th item, and $\hat{\mathbf{y}}_i$ is the predicted probability of the i-th item by the model. The cross-entropy loss function penalizes the model for making wrong predictions by increasing the loss value. The lower the cross-entropy loss, the better the model is at predicting the correct labels.

3.2.3.1 Hyperparameters

We train the model using the following hyperparameters:

- We use **30 epochs** for training, although we observe that the loss stabilizes in the last 10 epochs.
- We use a batch size of **100 sessions**, which allows us to leverage the parallelism of the

GPU[22].

- We set the number of **propagation steps** in the GNN layer to **2**, and naturally, the number of **recurrent propagation steps** in the GRU cell to **1**. We also experiment with using only 1 GNN propagation step in and find that it yields similar results.
- We use a hidden size $d \in \mathbb{R}^{1\times 100}$ for all the representations in the model, including the node representations, the session representation, and the item embeddings.

3.2.3.2 Optimization and Regularization

Initialization. We initialize all parameters using a Gaussian distribution with zero mean and a standard deviation of 0.1. This value is calculated by taking the inverse of the square root of the hidden size, as shown in the equation below:

$$\text{deviation} = \frac{1.0}{\sqrt{d}} \tag{3.14}$$

Optimization. We optimize the model using the mini-batch Adam optimizer, which is a variant of gradient descent that uses adaptive learning rates and momentum. We set the initial learning rate to 0.001 and decay it by 0.1 after every 3 epochs. We use a batch size of 100 sessions, which means that we update the parameters after processing 100 sessions in each iteration. The batch size affects the performance of the optimizer by balancing the speed and accuracy of the updates.

Regularization. As for the regularization, with use ridge regularization (L2) with a penalty of 10^{-5}. L2 regularization is usually applied only on kernel parameter matrices. But for our case, we applied it for all the learnable parameters of the model.

3.2.3.3 L-GGNN-ATT Variation

L-GGNN-ATT is our proposed variant of the SR-GNN model that adopts the Relevant Order Graph Formulation (see Sub-subsection 3.2.1.2). We define two or more sessions as *congruent* if they generate the same graph. A high frequency of congruent sessions in the dataset may result in many similar session graphs, which could potentially bias the model. We can infer that the longer the session, the lower the probability of congruence, thus diminishing the applicability of the Relevant Order Graph Formulation. Therefore, we raise that our proposed model is especially better for medium-length sessions, following Wang et al's survey [3] that suggested that medium-sized sessions are less prone to contain irrelevant interactions while preserving adequate contextual information.

[22]All of our training was done in a machine that has a Nvidia GeForce GTX 1080 with 2560 CUDA cores.

3.3 Implementation

3.3.1 Pandas Library

Figure 3.7: Pandas library for Python

Pandas[23] is a Python library that provides powerful and flexible data structures and tools for data analysis, manipulation, and cleaning. Pandas is built on top of numpy, which is a Python library that supports large, multi-dimensional arrays and matrices. Pandas allows us to work with "relational" or "labeled" data: tabular data or time series data. Pandas offers various features: handling missing data, grouping and aggregating data, merging and joining data sets, reshaping and pivoting data, slicing and indexing data, and reading and writing data from different sources. It is widely used in data science, machine learning, and statistics.

3.3.2 TensorFlow and Keras

Figure 3.8: TensorFlow logo

Figure 3.9: Keras logo

TensorFlow[24] and Keras[25] are two Python libraries that are used for machine learning and deep learning. TensorFlow is a platform that supports both high-level and low-level APIs, and it offers various features and tools for building, training, and deploying complex and advanced models and operations.

TensorFlow (see figure 3.8) can also run on multiple devices and platforms like GPUs, TPUs, mobile devices, and web browsers. Keras (see figure 3.9) is a high-level API that runs on top of TensorFlow and other frameworks, and it provides an easy-to-use, intuitive, and concise interface for creating and experimenting with neural networks. Keras covers every step of the machine

[23] https://pandas.pydata.org/
[24] https://www.tensorflow.org/
[25] https://keras.io/

learning workflow, from data processing to model evaluation to deployment. Keras is designed to reduce the cognitive load and enable fast prototyping for beginners and experts alike. Modules like `GRUCell` and `LSTMCell` help us implement RNNs without having to do it from scratch in a less efficient manner.

3.4 Datasets

3.4.1 YouChoose

The YouChoose dataset is a large-scale e-commerce dataset that was used for the RecSys Challenge 2015. The challenge was organized by YOOCHOOSE, a company that provides recommender system as a service to online retailers. The task was to predict what items a user intends to purchase, if any, given a click sequence performed during an activity session on the e-commerce website. The dataset contains about 9.2 million click events and 115,000 purchase events from more than 24,000 users over a period of six months. The dataset is publicly available on Kaggle[26] and GitHub[27].

3.4.2 DIGINETICA

This dataset was provided by DIGINETICA and its partners as part of the CIKM Cup 2016 Track 2[28], and it consists of anonymized search and browsing logs, product data, and anonymized transactions. The main goal of this dataset is to train models that can recommend products based on the customers' specific purchase, search, and browsing behaviors. The dataset includes anonymized user ids, hashed query phrases, hashed product descriptions and meta-data, pricing, clicks, and purchases extracted from user sessions in an e-commerce search engine log.

3.4.3 Dataset Preprocessing

We apply the same filtering criteria to both datasets: (1) removing sessions with only one item and items that occur less than five times. This results in 7,981,580 sessions and 37,483 items for the Yoochoose dataset, and 204,771 sessions and 43,097 items for the Diginetica dataset; (2) we create session subsequences and labels by splitting the input sequence. We use the sessions from the following days as the test set for Yoochoose, and the sessions from the following weeks as the test set for Diginetica. For time efficiency purposes, we will use only 1/64. After the preprocessing, the datasets' properties are as shown in table 3.1.

[26]https://www.kaggle.com/code/danofer/2015-recsys-challenge-starter
[27]https://github.com/RecoHut-Datasets/yoochoose
[28]https://competitions.codalab.org/competitions/11161#learn_the_details-overview]

Stats	YouChoose 1/64	Diginetica
Clicks	557,248	982,961
Training sessions	369,859	719,470
Testing sessions	55,898	60,858
Items	16,766	43,097
Avg. length	6.16	5.12

Table 3.1: YouChoose and Diginetica statistics

3.5 Performance

3.5.1 Evaluation Metrics

- **P@20 (Precision)** denotes the ratio of correctly recommended items among the first 20 items.

$$\text{P@20} = \frac{T_p^{(20)}}{T_p^{(20)} + Fp^{(20)}} \tag{3.15}$$

  $T_p^{(20)}$ is the number of *true positives* for the first 20 items, $F_p^{(20)}$ is the number of *false positives* for the first 20 items.

- **MRR@20 (Mean Reciprocal Rank)** is the mean of the inverse ranks of the correctly-recommended items. The inverse rank is zero if the rank is greater than 20. The MRR metric takes into account the order of recommendation ranking, where a high MRR value implies that correct recommendations are placed at the top of the list.

$$\text{MRR@20} = \frac{1}{20}\sum_{i=1}^{20}\frac{1}{rank_i} \tag{3.16}$$

3.5.2 Baselines

We compare the model with the following baselines:

- **BPR-MF** [65] optimizes a pairwise ranking objective function via stochastic gradient descent. This method uses matrix factorization to learn the latent factors of users and items from implicit feedback.

- **POP** and **S-POP** recommend the top-N frequent items in the training set and in the current session respectively. These methods are simple but effective for capturing the popularity of items.

- **Item-KNN** [66] recommends items similar to the previously clicked item in the session, where similarity is defined as the cosine similarity between the vector of sessions. This method is based on the assumption that similar items tend to be clicked together.

- **FPMC** [22] is a sequential prediction method based on markov chain. This method models

the transition probabilities between items in a session and incorporates user preferences using matrix factorization.

- **STAMP** [67] captures users' general interests of the current session and current interests of the last click. This method uses an attention network to learn the relevance between the last click and the previous clicks in a session.
- **NARM** [26] also employs RNNs but with an attention mechanism to capture the user's main purpose and sequential behavior. This method combines a global encoder and a local encoder to learn a hybrid representation of the session.
- **GRU4REC** [68] uses RNNs to model user sequences for the session-based recommendation. This method can capture the sequential dependencies and temporal dynamics of user behavior in a session.
- **SR-GNN** [5] uses GGNNs to model the complex transitions of items, which are challenging to be discovered by previous conventional sequential methods. Much like NARM, SR-GNN also uses an attention network to represent each session as the integration of the global preference and the current interest of that session.

3.5.3 Results and Discussion

3.5.3.1 Results

L-GGNN-ATT and SR-GNN have similar training losses, but L-GGNN-ATT's converges a bit faster than SR-GNN (see figure 3.10).

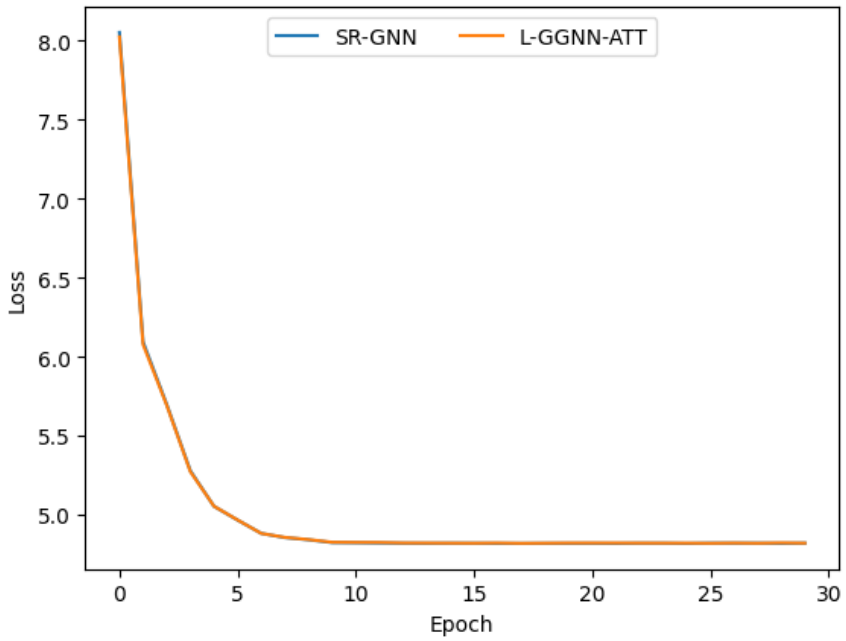

Figure 3.10: Training loss of both SR-GNN and L-GGNN-ATT for the DIGINETICA dataset

In table 3.2, methods based on neural networks, NARM and STAMP, achieve better results than the conventional methods, showing the effectiveness of applying deep learning in this domain. Short/long-term memory models, like GRU4REC and NARM, use recurrent units to capture a user's general interest while STAMP enhances the short-term memory by using the last-clicked item. These methods explicitly model the users' global behavioral preferences and consider transitions between users' previous actions and the next click, leading to better performance than these traditional methods. However, their performance is still lower than that of the SR-GNN model.

Compared with the state-of-art methods like NARM and STAMP, L-GGNN-ATT further considers transitions between items in a session and thereby models every session as a graph, which can capture more complex and implicit connections between user clicks. Whereas in NARM and GRU4REC, they explicitly model each user and obtain the user representations through separated session sequences, with possible interactive relationships between items ignored. Therefore, the model is more powerful to model session behavior.

Model	**YouChoose 1/64**		**Diginetica**	
	P@20	MRR@20	P@20	MRR@20
Item-KNN	51.60	21.81	35.75	11.57
BPR-MF	31.31	12.08	5.24	1.98
FPMC	45.62	15.01	26.53	6.95
GRU4REC	60.64	22.89	29.45	8.33
NARM	68.32	28.63	49.70	16.17
STAMP	68.74	29.67	45.64	14.32
SR-GNN	**70.57**	**30.94**	**50,73**	**17.59**
L-GGNN-ATT	**71.04**	**31.17**	**50,94**	**17.72**

Table 3.2: The model shows better performance than the state-of-the-art baselines

3.5.3.2 Discussion

Our proposed L-GGNN-ATT variation, which is SR-GNN with the Relevant Order Graph Formulation has a slightly better performance than its vanilla counterpart which uses the Edge Occurrence over Source Outdegree Ratio Normalization (see figure 3.11 and 3.12). The probability of sessions forming the same graph decreases as the sequence length increases. However, according to the SR-GNN paper, the model performs similarly irrespective of the average session size (short: length ≤ 5; long: length > 5). This suggests that the Relevant Order Graph Formulation only slightly enhances the model since the only requirement for a session graph is that the connections between nodes are nonzero for message propagation across the graph, and thus for global influence between items.

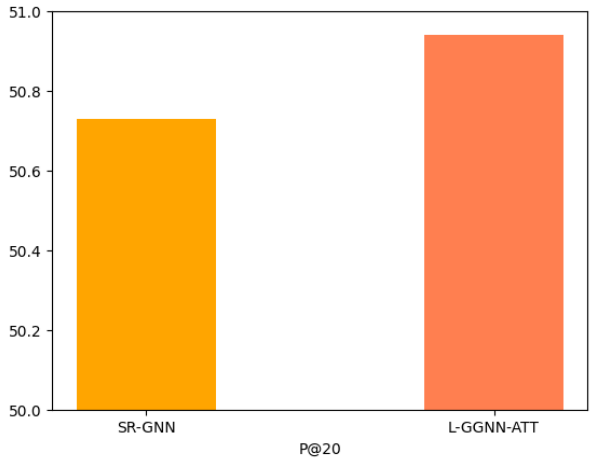

Figure 3.11: P@20 comparison between SR-GNN and L-GGNN-ATT for the DIGINETICA dataset

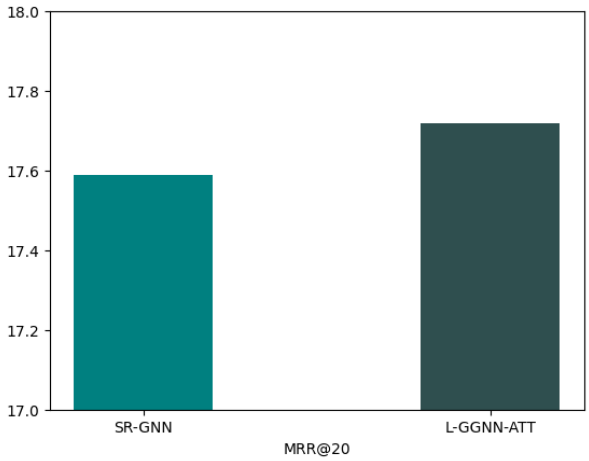

Figure 3.12: MRR@20 comparison between SR-GNN and L-GGNN-ATT for the DIGINETICA dataset

3.6 Conclusion

We proposed a state-of-the-art method for session-based recommendation based on gated graph neural networks (GGNNs), called L-GGNN-ATT. This method can effectively model the complex and dynamic user behavior in session data by mapping them to graphs and learning node and session representations without losing valuable session information. By using a soft-attention mechanism, this method can also capture the importance of different items in a session and their relevance to the last item. We showed how this deep learning approach can bring good results for session-based recommendation by comparing it with several baselines on two real-world datasets. This method provides more accurate and personalized recommendations for anonymous and dynamic users.

CHAPTER:
General Conclusion

General Conclusion

Summary

This study aimed to investigate the application of GGNNs for session-based recommender systems. The main research questions were:

1. How can GGNNs be used to model user sessions and capture the sequential dependencies among items?
2. How does the GGNN-based model compare with the state-of-the-art baselines in terms of recommendation accuracy and diversity?
3. What are the advantages and limitations of using GGNNs for SBRSs?

To answer these questions, the study was divided into three main chapters. The first chapter provided a theoretical background on recommender systems, sequential recommender systems, and session-based recommender systems, highlighting their challenges, opportunities, and applications.

The second chapter introduced the concepts of artificial intelligence, machine learning, deep learning, and graph neural networks, with a focus on GGNNs as a novel and powerful architecture for learning from graph-structured data.

The third chapter presented the practical implementation of the GGNN-based model for SBRSs, describing the evaluation metrics, datasets, baselines, and experimental results. A comparative analysis showed that the GGNN-based model outperformed the baselines, demonstrating its effectiveness and robustness for SBRSs. A discussion of the results revealed that the GGNN-based model was able to capture the complex and dynamic patterns of user behavior, leverage the rich information embedded in the item graph, and generate diverse and relevant recommendations.

However, some limitations were also identified, such as the computational complexity, the sensitivity to hyperparameters, and the lack of interpretability of the GGNN-based model. We conclude that GGNNs are a promising technique for SBRSs, offering a novel and flexible way of modeling user sessions and generating recommendations. Nonetheless, more research is needed to address the existing challenges and explore the potential extensions of GGNNs for SBRSs.

Directions for Future Research

Although GGNNs have shown promising results for SBRS, there are still many open questions and challenges that need to be addressed to fully exploit their potential and advance the state of the art. We suggest some possible avenues of research that can be pursued to further investigate or improve SBRS using GGNNs.

- Incorporating additional features or side information into the item graph, such as item attributes or user feedback.

- Developing hybrid models that combine GGNNs with other methods, such as collaborative filtering or matrix factorization.
- Developing novel attention mechanisms or aggregation functions that can capture the long-term and short-term preferences of users and the semantic relations among items in SBRS using GGNNs.
- Evaluating the robustness and generalization ability of SBRS using GGNNs under different settings and challenges like data sparsity, cold start, or data drift[29].

[29]Data drift, also known as covariate shift, occurs when the distribution of the input data changes over time.

References

[1] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide deep learning for recommender systems, 2016.

[2] Zhenhua Dong, Zhe Wang, Jun Xu, Ruiming Tang, and Jirong Wen. A brief history of recommender systems, 2022.

[3] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet Orgun, and Defu Lian. A survey on session-based recommender systems, 2021.

[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021. doi: 10.1109/tnnls.2020.2978386. URL https://doi.org/10.1109%2Ftnnls.2020.2978386.

[5] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based Recommendation with Graph Neural Networks. In Pascal Van Hentenryck and Zhi-Hua Zhou, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, volume 33 of *AAAI '19*, pages 346–353, July 2019. doi: 10.1609/aaai.v33i01.3301346. URL https://aaai.org/ojs/index.php/AAAI/article/view/3804.

[6] Wei Chen, Zhendong Niu, Xiangyu Zhao, and Yi Li. A hybrid recommendation algorithm adapted in e-learning environments. *World Wide Web*, 17(2):271–284, Mar 2014. ISSN 1573-1413. doi: 10.1007/s11280-012-0187-z. URL https://doi.org/10.1007/s11280-012-0187-z.

[7] Soo-Yeon Jeong and Young-Kuk Kim. Deep learning-based context-aware recommender system considering contextual features. *Applied Sciences*, 12(1), 2022. ISSN 2076-3417. doi: 10.3390/app12010045. URL https://www.mdpi.com/2076-3417/12/1/45.

[8] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Orgun. Sequential recommender systems: Challenges, progress and prospects. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6332–6338. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/883. URL https://doi.org/10.24963/ijcai.2019/883.

[9] Markos Aivazoglou, Antonios O. Roussos, Dionisis Margaris, Costas Vassilakis, Sotiris Ioannidis, Jason Polakis, and Dimitris Spiliotopoulos. A fine-grained social network recommender system. *Social Network Analysis and Mining*, 10(1):8, Dec 2019. ISSN 1869-5469. doi: 10.1007/s13278-019-0621-7. URL https://doi.org/10.1007/s13278-019-0621-7.

[10] Imran Hossain, Md Aminul Haque Palash, Anika Tabassum Sejuty, Noor A Tanjim, MD Abdullah AL Nasim, Sarwar Saif, Abu Bokor Suraj, Md Mahim Anjum Haque, and Nazmul Karim. A survey of recommender system techniques and the ecommerce domain, 2023.

[11] Prem Melville and Vikas Sindhwani. *Recommender Systems*, pages 829–838. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_705. URL https://doi.org/10.1007/978-0-387-30164-8_705.

[12] Haoren Zhu, Hao Ge, Xiaodong Gu, Pengfei Zhao, and Dik Lun Lee. Influential recommender system, 2022.

[13] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016. ISBN 978-3-319-29659-3.

[14] Fast Forward Cloudera, Jan 2021. URL https://session-based-recommenders.fastforwardlabs.com/.

[15] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, pages 257–297. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_8. URL https://doi.org/10.1007/978-0-387-85820-3_8.

[16] Maurizio Morisio Erion Çano. Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 21(6):1487–1524, nov 2017. doi: 10.3233/ida-163209. URL https://doi.org/10.3233%2Fida-163209.

[17] Gediminas Adomavicius and Alexander Tuzhilin. *Context-Aware Recommender Systems*, pages 217–253. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_7. URL https://doi.org/10.1007/978-0-387-85820-3_7.

[18] Amit Livne, Moshe Unger, Bracha Shapira, and Lior Rokach. Deep context-aware recommender system utilizing sequential latent context, 2020.

[19] Dietmar Jannach, Malte Ludewig, and Lukas Lerche. Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts. *User Modeling and User-Adapted Interaction*, 27, 12 2017. doi: 10.1007/s11257-017-9194-1.

[20] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, may 2000. ISSN 0163-5808. doi: 10.1145/335191.335372. URL https://doi.org/10.1145/335191.335372.

[21] Ghim-Eng Yap, Xiao-Li Li, and Philip S. Yu. Effective next-items recommendation via personalized sequential pattern mining. In Sang-goo Lee, Zhiyong Peng, Xiaofang Zhou, Yang-Sae Moon, Rainer Unland, and Jaesoo Yoo, editors, *Database Systems for Advanced Applications*, pages 48–64, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29035-0.

[22] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 811–820, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772773. URL https://doi.org/10.1145/1772690.1772773.

[23] F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999. doi: 10.1049/cp:19991218.

[24] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

[25] Fajie Yuan, Xiangnan He, Guibing Guo, Zhezhao Xu, Jian Xiong, and Xiuqiang He. Modeling

the past and future contexts for session-based recommendation. *CoRR*, abs/1906.04473, 2019. URL http://arxiv.org/abs/1906.04473.

[26] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, and Jun Ma. Neural attentive session-based recommendation, 2017.

[27] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2017.

[28] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[29] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[31] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Comput. Surv.*, 54(6), jul 2021. ISSN 0360-0300. doi: 10.1145/3457607. URL https://doi.org/10.1145/3457607.

[32] P. E. Miller. Predictive abilities of machine learning techniques may be limited by dataset characteristics: Insights from the unos database. *Journal of cardiac failure*, 2019.

[33] P. Jonathon Phillips, Carina Hahn, Peter Fontana, Amy Yates, Kristen K. Greene, David Broniatowski, and Mark A. Przybocki. Four principles of explainable artificial intelligence, 2021-09-29 04:09:00 2021. URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=933399.

[34] T.L. Fine. *Feedforward Neural Network Methodology*. Information Science and Statistics. Springer New York, 2006. ISBN 9780387226491. URL https://books.google.dz/books?id=s-PlBwAAQBAJ.

[35] Derya Soydaner. A comparison of optimization algorithms for deep learning. *CoRR*, abs/2007.14166, 2020. URL https://arxiv.org/abs/2007.14166.

[36] P. Bühlmann and S. van de Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Series in Statistics. Springer Berlin Heidelberg, 2011. ISBN 9783642201912. URL https://books.google.dz/books?id=lTvHNAEACAAJ.

[37] Siddharth Sharma. Activation functions in neural networks. 2020.

[38] F.M. Salem. *Recurrent Neural Networks: From Simple to Gated Architectures*. Springer International Publishing, 2022. ISBN 9783030899295. URL https://books.google.dz/books?id=bJpXEAAAQBAJ.

[39] S. Sharma. An adaptive sigmoidal activation function cascading neural networks. 2011.

[40] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969. doi: 10.1109/TSSC.1969.300225.

[41] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[42] Simon Kornblith. What's in a loss function for image classification? *ArXiv*, 2020.

[43] Like Hui. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. *ArXiv*, 2020.

[44] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci U S A*, 79(8):2554–2558, April 1982.

[45] Phong Le. Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive lstms. 2016.

[46] Niru Maheswaranathan. How recurrent networks implement contextual processing in sentiment analysis. *ArXiv*, 2020.

[47] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.03.091. URL https://www.sciencedirect.com/science/article/pii/S092523122100477X.

[48] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention, 2016.

[49] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

[50] Yujia Liu, Kang Zeng, Haiyang Wang, Xin Song, and Bin Zhou. Content matters: A gnn-based model combined with text semantics for social network cascade prediction. In Kamal Karlapalem, Hong Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty, editors, *Advances in Knowledge Discovery and Data Mining*, pages 728–740, Cham, 2021. Springer International Publishing. ISBN 978-3-030-75762-5.

[51] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[52] Maciej Krzywda, Szymon Lukasik, and Amir H. Gandomi. Graph neural networks in computer vision - architectures, datasets and common approaches. In *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2022. doi: 10.1109/ijcnn55064.2022.9892658. URL https://doi.org/10.11092Fijcnn55064.2022.9892658.

[53] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. Graph neural networks for natural language processing: A survey, 2022.

[54] Petar Veličković. Message passing all the way up, 2022.

[55] Giannis Nikolentzos, Antoine J. P. Tixier, and Michalis Vazirgiannis. Message passing attention networks for document understanding, 2019.

[56] Eric Alcaide. Improving graph property prediction with generalized readout functions, 2020.

[57] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *J Comput Aided Mol Des*, 30(8):595–608, August 2016.

[58] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data, 2015.

[59] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks, 2022.

[60] Florian Grötschla, Joël Mathys, and Roger Wattenhofer. Learning graph algorithms with recurrent graph neural networks, 2022.

[61] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[62] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[63] Daniel Selsam, Matthew Lamm, Benedikt Bunz, Percy Liang, Leonardo Moura, and David Dill. Learning a sat solver from single-bit supervision. 02 2018.

[64] Tianwen Chen and Raymond Chi-Wing Wong. Handling information loss of graph neural networks for session-based recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1172–1180, 2020.

[65] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012.

[66] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133480. doi: 10.1145/371920.372071. URL https://doi.org/10.1145/371920.372071.

[67] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: Short-term attention/memory priority model for session-based recommendation. pages 1831–1839, 07 2018. doi: 10.1145/3219819.3219950.

[68] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks, 2016.