EXPLORING DEEP LEARNING TECHNIQUES TO TACKLE THE SPARSITY PROBLEM IN RECOMMENDER SYSTEMS

A Dissertation presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

MESHAL ALFARHOOD

Prof. Jianlin Cheng, Dissertation Supervisor

DECEMBER 2020

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

EXPLORING DEEP LEARNING TECHNIQUES TO TACKLE THE SPARSITY PROBLEM IN RECOMMENDER SYSTEMS

presented by Meshal Alfarhood,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Prof. Jianlin Cheng

Prof. Dong Xu

Prof. Jeffrey Uhlmann

Prof. Stephen Montgomery-Smith

DEDICATION

I dedicate this dissertation to my parents, Dawood and Haya, for their endless support and unconditional love throughout my whole life.

ACKNOWLEDGMENTS

First, I would like to express my deepest appreciation to my advisor and committee chair, Prof. Jianlin Cheng, for his guidance, patience, and motivation during my doctoral studies. Under his supervision, I had the opportunity to explore several directions of my research that enriches my learning knowledge, and develops my technical, theoretical and personal abilities. Second, I would like to thank the rest of my dissertation committee: Prof. Dong Xu, Prof. Jeffrey Uhlmann, and Prof. Stephen Montgomery-Smith for their valuable feedback and insightful comments.

Also, I am extremely thankful to my sponsor, King Saud University, for their generous scholarship that covers my expenses throughout my master and PhD degrees. I also extend my gratitude to the University of Missouri for giving me this opportunity to be one of its alumni family.

A special appreciation goes to my older brother, Dr. Sultan. His support and encouragement were one of the reasons what made this dissertation possible.

Finally, I would like to extend my sincere and obligation to my family back home for the prayers, the long-distance support, and for believing in me. I am also grateful to my friends in Columbia, MO for the great moments we had spent together. Without all of them, I would have not reached at this point where I am writing this final part of my dissertation!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the inception of e-commerce in the early twenty-first century, people's lifestyles have drastically changed. People today tend to do many of their daily routines online, such as shopping, reading the news, and watching movies. Nevertheless, consumers often face difficulties while exploring related items such as new fashion trends because they are not aware of their existence due to the overwhelming amount of information available online. This phenomenon is widely known as "information overload". Therefore, recommender systems (RSs) are a critical solution for helping users make decisions when there are lots of choices. RSs have been integrated into and have become an essential part of every website due to their effectiveness in increasing customer interactions, attracting new customers, and growing business revenue.

Machine learning, and deep learning (DL) in particular, have achieved a great success in resolving various computer science problems. Generally, DL-based approaches have enhanced performance remarkably compared with traditional approaches. Specifically, DL-based approaches have become the state-of-the-art techniques in RSs. Therefore, in this dissertation, three DL-based contributions are presented to address the natural data sparsity problem in RSs: (1) *DeepHCF*, a deep-hybrid, collaborative-filtering model that trains two deep models via joint training for rating prediction tasks; (2) *CATA*, a collaborative attentive autoencoder that integrates the attention mechanism to enhance the recommendation quality for ranking prediction tasks; and (3) *CATA++*, an extended version of CATA that employs a dual attentive autoencoder to leverage more of the item's content. All proposed models have gone through comprehensive experiments to evaluate their performance against state-of-the-art models using real-world datasets. Our experimental results show the superiority of our models over state-of-the-art models according to various evaluation metrics.

Chapter 1

Introduction

1.1 Recommender systems

The amount of data created in the last few years is overwhelming. Interestingly, more than 90% of data has been created in only the last two years [1]. The term "information overload" has gained popularity recently, describing the difficulties people face in finding what they want from such a huge volume of available information. There are two areas in computer science that are concerned with providing information processing techniques to address the information overload, i.e., information retrieval (IR) and recommender systems (RSs). In IR systems (e.g., the Google search engine[1]), users submit requests for what they want, and the search engine tries to find the optimal corresponding results for those requests. On the other hand, RSs obtain users' preferences first and then provide item recommendations based on those preferences without users explicitly requesting them [2].

[1] www.google.com

1.1.1 Motivation

Recommender systems are everywhere today. Most of commercial and social websites use recommendation engines to show relevant items to their users. This process usually increases user activity and interactions with their websites. RSs have been incorporated into different recommendation tasks such as recommending what videos to watch [3], scientific papers to read [4], places to visit [5], and songs to which to listen [6]. According to YouTube [7], 60% of the videos watched on YouTube[2] come from its homepage recommendations. Also, 80% of what Netflix[3] users watch comes from its recommender system as well, while the remaining 20% comes from users' searches [8]. Moreover, a company like Amazon[4] use recommender systems to increase its profit by 35% [9]. Finally, Google has successfully increased the click-through rate (CTR) in its news service[5] by 38% using recommendations [10]. Therefore, the ongoing research in this field is important for both academia and industry, and holds the key for some businesses to engage effectively with and serve their users.

Over the last few decades, a lot of effort has been made by both academia and industry in proposing new ideas and solutions for RSs, which ultimately help service providers in adopting such models into their system architecture. RSs research has evolved remarkably following the Netflix prize competition[6] in 2006, where the company offered one million dollars for any team that could improve their recommendation accuracy by 10%. Therefore, our main focus in this dissertation is to define research challenges facing recommender systems, study the most popular works and how they address such challenges, and then report our main contributions regarding this matter. We now define what the "recommendation problem" is in the following

[2] www.youtube.com
[3] www.netflix.com
[4] www.amazon.com
[5] www.news.google.com
[6] www.netflixprize.com

section.

1.1.2 Problem definition

There are four primary elements involved in defining the recommendation problem. Users (e.g., customers, authors, listeners) and items (e.g., movies, books, songs) are the two main entities needed to build any commercial system. Let U be the set of users and I be the set of items, such that $U = \{u_1, u_2, .., u_n\}$ and $I = \{i_1, i_2, .., i_m\}$. Items are usually points of interest for users, such that users interact with (e.g., buy, watch, like) items. The user-item interactions, which are considered as the third element in RSs, can be represented as a scoring function ($R_{ui} = score$), such that items with higher score values more likely match user preferences. Finally, additional information that can be collected from either users (e.g., age, country, sex) or items (e.g., movies' reviews, books' content, songs' lyrics) can be seen as the fourth element of RSs.

In order for recommender systems to generate recommendations, they first need to collect the user-item interaction data and feed this data into the system for training. This user-item interaction data can be either collected in an explicit or implicit manner. In explicit data, users directly express their opinion about an item using the rating system to show how much they like that item. User ratings usually vary from one star to five stars with five being very interested and one being not interested. This type of data is very useful and reliable due to the fact that it represents users' actual feelings about those items. However, user ratings are occasionally not available due to the difficulty of obtaining users' explicit opinions. In this case, implicit feedback can be obtained indirectly from user behavior such as user clicks, bookmarks, or the time spent viewing an item. For example, if a user listens to a song ten times in the last

two days, he or she most likely likes this song. Thus, implicit data is more prevalent and easier to collect, but it is generally less reliable than explicit data. Also, all the observed interactions in implicit data constitute positive feedback, where negative feedback is missing. This problem is also defined as the "one-class problem".

There are multiple previous works that have aimed to address the one-class problem. A simple solution is to treat all missing data as negative feedback. However, this is not true because the missing (unobserved) interaction could be positive if the user is aware of the item existence. Therefore, using this strategy to build a model might result in a misleading model due to the wrong assumption made at the beginning. On the contrary, if we treat all missing data as unobserved data without considering negative feedback, the corresponding trained model is more likely useless since it is only trained on positive data. As a result, sampling negative feedback from positive feedback is one practical solution for this problem, which has been proposed by [11]. In addition, Weighted Regularized Matrix Factorization (WRMF) [12] is another proposed solution that introduces a confidence variable that works as a weight to measure how likely a user is to like an item. This approach has been described thoroughly in Section 4.3.1.

In general, the user-item interaction data is usually represented by a matrix $R \in \mathbb{R}^{n \times m}$, where n represents the number of users and m represents the number of items. R_{nm} shows the rating value of $User_n$ for $Item_m$. As stated before, the rating values usually range from one to five for explicit feedback. On the other hand, the recommendation problem with implicit data is usually formulated as follows:

$$R_{nm} = \begin{cases} 1, & \text{if there is user-item interaction} \\ 0, & \text{if otherwise} \end{cases} \tag{1.1}$$

Table 1.1: The representations of the user-item interaction matrices for (a) explicit and (b) implicit feedback data.

(a) Explicit feedback

	$\mathbf{item}_0$	$\mathbf{item}_1$	$\mathbf{item}_2$	**...**	$\mathbf{item}_m$
$\mathbf{user}_0$	2	3		...	
$\mathbf{user}_1$	5		1	...	4
$\mathbf{user}_2$		3		...	5
...	...	...	...	...	...
$\mathbf{user}_n$	4	2	5	...	

(b) Implicit feedback

	$\mathbf{item}_0$	$\mathbf{item}_1$	$\mathbf{item}_2$	**...**	$\mathbf{item}_m$
$\mathbf{user}_0$	1	1	0	...	0
$\mathbf{user}_1$	1	0	1	...	1
$\mathbf{user}_2$	0	1	0	...	1
...	...	...	...	...	...
$\mathbf{user}_n$	1	1	1	...	0

where the values of ones in implicit feedback represent all the positive feedback. However, it is important to notice that a value of zero does not imply always negative feedback. It can be that users are not aware of the existence of those items. Examples of what matrix R looks like in both cases are shown in Table 1.1.

Matrix R is usually highly imbalanced, where the number of the observed interactions is much less than the number of the unobserved interactions. In other words, matrix R is very sparse, meaning that users only interact either explicitly or implicitly with a very small number of items compared to the total number of items in matrix R. This sparsity problem is one frequent problem in RSs, which brings a real challenge for any proposed model to have the capability to provide effective personalized recommendations in this context. The sparsity of the user-item matrix can be measured as the number of the interactions between users and items (e.g. the

number of all observed ratings in explicit data, and the number of all ones in implicit data) divided by all the possible interactions in this matrix (i.e., the number of users times the number of items).

1.2 Dissertation contributions and outline

This dissertation is organized into six chapters in the following order:

- **Chapter 1** shows the motivation of this dissertation and defines the recommendation problem.

- **Chapter 2** describes recommender system techniques and their different evaluation metrics, and then illustrates deep learning techniques and their applications in recommender systems.

- **Chapter 3** presents DeepHCF, a deep-hybrid collaborative-filtering model that is designed and evaluated for a rating prediction task with explicit feedback data. The main content of this chapter is part of this publication:
  Alfarhood, M. *and Cheng, J. (2018). DeepHCF: A deep learning based hybrid collaborative filtering approach for recommendation systems. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 89-96). IEEE.* [13].

- **Chapter 4** presents CATA, a collaborative attentive autoencoder that is designed and evaluated for a ranking prediction task with implicit feedback data. The main content of this chapter is part of the following publications:
  Alfarhood, M. *and Cheng, J. (2019). Collaborative Attentive Autoencoder for Scientific Article Recommendation. In 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA) (pp. 168-174). IEEE.*

[14].

Alfarhood, M. *and Cheng, J. (2021). Deep Learning-Based Recommender System. In Deep Learning Applications, Volume 2. Advances in Intelligent Systems and Computing, Vol 1232, (pp. 1-23). Springer, Singapore.* [15].

- **Chapter 5** presents CATA++, a dual attentive autoencoder model. It is an improvement over our previous model, CATA, by leveraging articles contents via double deep models. The main content of this chapter is part of this publication:

  Alfarhood, M *and Cheng, J. (2020). CATA++: A Collaborative Dual Attentive Autoencoder Method for Recommending Scientific Articles. in IEEE Access, vol. 8 (pp. 183633-183648). IEEE.* [16].

- **Chapter 6** concludes this dissertation and suggests potential directions for future work.

Chapter 2

Preliminaries and Related Work

This chapter is organized into three main sections. We first describe the three well-known categories of recommender systems, i.e., collaborative filtering approaches, content-based filtering approaches, and hybrid approaches. We provide existing related work for each category. Afterward, we illustrate the different evaluation metrics used in recommender systems according to the data type and what the model is trying to measure. Finally, we go through the deep learning techniques and their literature with regard to recommendations.

2.1 Recommendation techniques

Recommender systems generally are divided into two types, namely personalized RSs and non-personalized RSs. Personalized RSs are based on the preferences of users, while non-personalized RSs involve no personal preferences in their recommendations. Recommending the most common or highly rated items in a training dataset is an example of a non-personalized RSs, where all users get identical recommendations

[17]. Although non-personalized RSs might be useful and effective in some cases, they are not generally within the scope of RSs research. Personalized RSs, on the other hand, have three known models: collaborative filtering (CF), content-based filtering (CBF), and hybrid models.

2.1.1 Collaborative filtering techniques

Collaborative filtering (CF) models (e.g., Singular Value Decomposition (SVD) [18]) focus on users' histories, such that users with similar past behaviors tend to have similar future tastes. For instance, if $User_a$ and $User_b$ are similar to each other, they are more likely to agree on an item in the future. Matrix Factorization (MF) has been one of the most popular CF techniques for many years and has been widely used in the recommendation literature. Many proposed models are enhanced versions of MF, such as Probabilistic Matrix Factorization (PMF) [19], Bayesian Probabilistic Matrix Factorization (BPMF) [20], Sparse Probabilistic Matrix Factorization (SPMF) [21], and Weighted Regularized Matrix Factorization (WRMF) [12]. MF is described in depth in Section 4.3.1.

CF models are classified into two categories based on how models are trained, i.e., memory-based CF and model-based CF [22]. Memory-based CF models (a.k.a. neighborhood-based models) [23, 24] work by finding similarities among users (user-based CF) or items (item-based CF) directly from the user-item ratings matrix, and then predicting a user's rating based on the average ratings of the k-nearest neighbors. Unlike CBF models, similarities among items here are higher when two items in the system are rated by multiple users in the same manner. For instance, user-based CF models are reflected in the following popular phrasing used on many websites: "*users similar to you also bought this item*". Popular phrasing for item-based CF

models is: "*users who bought this item also bought this item*". Thus, calculating the similarity is the base of this kind of approach. Cosine-similarity is one common similarity measurement method to measure the cosine of the angle between two users vectors as follows:

$$cos(A, B) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.1}$$

where $\cdot$ indicates the dot product between the two vectors and $\|A\|$ is the norm of that vector.

Another common similarity measurement method is the Pearson correlation similarity, where its coefficient, ρ, is calculated as:

$$\rho_{A,B} = \frac{cov(A, B)}{\sigma_A \sigma_B} = \frac{\sum_{i=1}^{n}(a_i - \overline{a})(b_i - \overline{b})}{\sqrt{\sum_{i=1}^{n}(a_i - \overline{a})^2} \sqrt{\sum_{i=1}^{n}(b_i - \overline{b})^2}} \tag{2.2}$$

where *cov* is the covariance, and σ is the standard deviation.

Alternatively, Mean Squared Difference (MSD) and Mean Absolute Difference (MAD) are other popular similarity functions used to measure how close two users or items are to each other. They can be calculated as follows:

$$\begin{aligned} MSD(A, B) &= \frac{1}{|C|} \sum_{i \in C} (A_i - B_i)^2 \\ MAD(A, B) &= \frac{1}{|C|} \sum_{i \in C} |A_i - B_i| \end{aligned} \tag{2.3}$$

where C is the set of items that are rated by both users A and B.

In contrast, model-based CF models (a.k.a. latent factor models) apply machine learning techniques to model users and items as vectors in the latent space. Matrix factorization-based models and neural networks are the most popular examples of

this approach. Model-based CF approaches are generally more popular than memory-based CF approaches, and they also handle the matrix sparsity much better. However, they are obviously slower in the training phase than the memory-based methods since they need to build and train the model first. On the contrary, they are faster in the testing phase once the model is trained since the memory-based methods need to run the process over again each time they need to do a testing.

However, CF models generally rely only on users' past ratings in their learning processes and do not consider other auxiliary information validated later to improve the quality of recommendations. For that reason, CF models suffer when there are new users or new items arrive to the system. These models are neither able to give recommendations to new users because there is not enough information about those users, nor are they able to recommend new items to existing users because those items have not seen before by any user [25]. This particular issue is known as the "cold-start problem". In addition, the performance of CF models drops substantially when users or items have insufficient amounts of feedback data. This problem is known as the "data sparsity problem".

2.1.2 Content-based filtering techniques

Content-based filtering (CBF) models, on the other hand, work by learning an item's features from that item's information and offers recommendations based on the similarities among items, rather than the similarities among users like CF models do. Item similarities here are calculated from the item's information, such as the item title and genre. For example, if $User_a$ likes $Item_b$, then CBF models look for other items that have characteristics similar to $Item_b$ to make their recommendations. Unlike CF models that need other users' ratings, CBF models are user-independent [26].

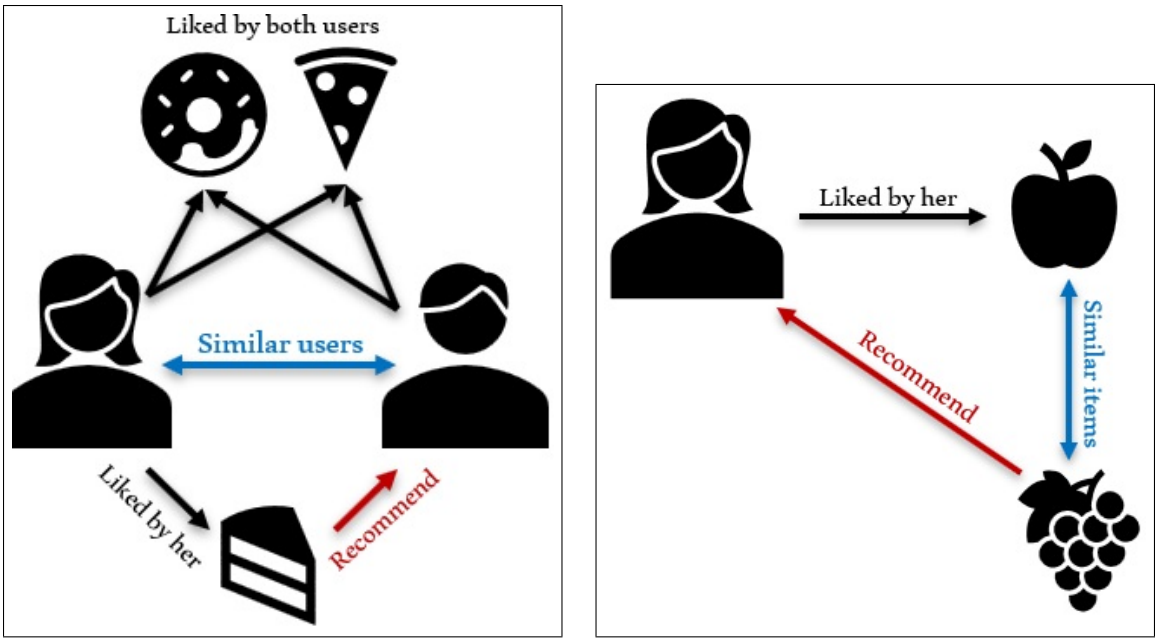
Furthermore, new items can be recommended to existing users because the recommendation here is based on item features and not user feedback. Figure 2.1 shows a basic example of the difference between CF and CBF approaches in how they generate recommendations.

However, CBF models have two major challenges, i.e., the limited content analysis problem and the overspecialization problem. The first problem refers to the difficulty of extracting information from items because they are not always available, while the second one appears when a RS only gives item recommendations similar to the items users liked before [27]. Consequently, CBF models lack novelty and diversity in their recommendations [28]. Generally, they have lower performance than CF models.

DeepMusic [29] is an example of a CBF model that relies only on item information and neglects user ratings. It learns latent factors only from the audio signals to recommend musics. In addition, LIBRA [30] is a book RS that extracts book information from Amazon web pages and then gives recommendations using a bag-of-words naive Bayesian text classifier. Finally, INTIMATE [31] is a movie RS that gives recommendations based only on movies' synopses using text categorization techniques.

2.1.3 Hybrid techniques

More recently, much effort has been put forth to include an item's information along with the user's ratings data via topic modeling [4, 32, 33]. For example, Collaborative Topic Regression (CTR) [4] is composed of Probabilistic Matrix Factorization (PMF) and Latent Dirichlet Allocation (LDA) to utilize both a user's ratings and an item's reviews to learn their latent features. By doing this, the natural sparsity problem can be alleviated, and these kinds of approaches are called "hybrid models". Hybrid models are combinations between CF and CBF approaches to take advantage of both



(a) Collaborative filtering approach (b) Content-based filtering approach

Figure 2.1: A basic example to show the difference in recommendations between (a) collaborative filtering and (b) content-based filtering approaches.

approaches' positive aspects and to overcome some of the issues each possess. For example, the sparsity problem and the cold start problem are addressed by this hybrid approach. However, this type of approach increases the complexity of the model and usually needs more time to train [25]. In this dissertation, we propose three hybrid approaches that use both a user's feedback history and an item's auxiliary information.

Generally, hybrid models are divided into two sub-categories according to how models are trained: loosely coupled models and tightly coupled models [34]. Loosely coupled models train CF and CBF models separately, like ensembles, and then determine the final score based on the scores of the two separated models. On the other hand, the tightly coupled models train both CF and CBF models jointly. In joint training, both models cooperate with each other to calculate the final score under the same loss function.

2.2 Recommendation evaluation metrics

Recommendation tasks can be classified into two different tasks based on the type of input data. They can be either rating prediction or ranking prediction tasks [35]. In rating predictions, models use explicit feedback to train their model to predict a future rating a user would give to an item. On the other hand, ranking predictions (a.k.a. top-K recommendation) work by suggesting a list of items to a user and ranking them based on user preferences. Implicit feedback is widely used by ranking prediction models [12, 11, 36]. Ranking prediction tasks are more popular in the literature than rating prediction tasks due to the ease of collecting implicit data [35]. There are different metrics to evaluate each recommendation task, which are described in the following sections.

2.2.1 Rating accuracy metrics

Since the rating predication task seeks to predict ratings users would give to items not yet seen, the accuracy computed here by calculating the error between the predicted ratings and the actual ratings. There are three standard error metrics to compute accuracy: Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). MSE and RMSE give greater penalty to larger errors by squaring the error, as illustrated in the following equations:

$$\begin{aligned} MAE &= \frac{1}{n}\sum_{i=1}^{n}|r_i - \hat{r}_i| \\ MSE &= \frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2 \\ RMSE &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2} \end{aligned} \tag{2.4}$$

where r_i is the actual rating, $\hat{r}_i$ is the predicted rating, and n is the number of samples.

2.2.2 Classification accuracy metrics

On the other hand, the ranking prediction task seeks to recommend K items to each user. The common way to evaluate this case is by computing how many correct or relevant items within the top-K recommended items. Precision and recall are the most popular methods to do this. Precision measures the ratio between the number of correct recommendations and the number of all recommended items, while recall measures the ratio between the number of correct recommendations and the number of all relevant items in the testing dataset, as follows:

$$\begin{aligned} precision &= \frac{|\text{Relevant items} \cap \text{Recommended items}|}{|\text{Recommended items}|} \\ recall &= \frac{|\text{Relevant items} \cap \text{Recommended items}|}{|\text{Relevant items}|} \end{aligned} \tag{2.5}$$

There is a trade-off between the recall and precision metrics, such that increasing the number of recommended items never decreases the recall value, but generally decreases the precision value. Therefore, the F1 score combines both metrics together and works as a harmonic mean of the recall and precision, as the following:

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.6}$$

2.2.3 Ranking metrics

Yet, the previous metrics do not take into account the ranking of the recommended items. They treat all the K recommended items the same. However, it is often important for some systems to reward items on the top of the list more than others. Thus,

there are three popular ranking metrics for this situation. First, Mean Reciprocal Rank (MRR) evaluates the ranking of the first correct recommendation as:

$$MRR = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{rank_i} \tag{2.7}$$

where $|U|$ is the total number of users, i is the rank of the recommended items, and $rank_i$ is the rank of the first relevant item within the top-K recommendations.

Second, Mean Average Precision (MAP) shows the average precision for all relevant items. It is different than the original precision metric (Equation 2.5), such that MAP has higher final values for items at the top of the recommended item list than at the bottom of the list. The original precision metric gives similar results as long as they are within the recommended list. This is shown by the following equation:

$$MAP = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{\sum_{i=1}^{n} precision(i) \cdot rel(i)}{|R|} \tag{2.8}$$

where R is the set of all relevant items, n is the number of recommended items, $precision(i)$ is the precision value at rank i, and $rel(i)$ is an indicator function where it returns a one if the item at rank i is a relevant item and a zero otherwise.

Third, Discounted Cumulative Gain (DCG) is another popular ranking metric where the score of each recommended item is reduced logarithmically, proportional to the position of that item. Unlike MAP, which works only with binary outcomes, DCG can work also with non-binary outcomes in the following manner:

$$DCG = \frac{1}{|U|} \sum_{u=1}^{|U|} \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{2.9}$$

In addition, normalized Discounted Cumulative Gain (nDCG) normalizes DCG by

dividing its result by the ideal DCG (IDCG) where all the correct recommendations in IDCG appear at the top of the list. Thus, the value of nDCG now ranges between zero and one, as:

$$nDCG = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{DCG}{IDCG} \tag{2.10}$$

2.3 Deep learning in recommendations

The amount of data that has been created lately grows exponentially from the years before. Because of that, the term "big data" gets more recognition, with this term describing the massive body of unstructured data generated that requires further analysis. Machine learning, and deep learning in particular, have gained increasing attention in recent years due to the way they enhance how we process big data, and also for their capability to model complicated data such as texts and images. Recently, deep learning has become a promising tool for different data domains due to the increase in computational resources like GPUs, the exponential growth of available data, and improved algorithms. For instance, deep learning models have been successfully applied in computer vision, speech recognition, and natural language processing (NLP) problems.

The application of deep learning to recommender systems is recent. Even though traditional recommendation approaches have achieved a great success in the last two decades, they still have shortcomings in accurately modeling complex (e.g., non-linear) relationships between users and items. Alternatively, deep neural networks are universal function approximators that are capable of modeling any continuous function. One of the first works that applies deep learning concept for collaborative filtering is Restricted Boltzmann Machine (RBM) [37]. It performs inference by us-

ing a two-layer architecture to learn users' tastes from their histories. However, this model is not deep enough (two layers only) to capture the features of users and items, and furthermore, it does not employ other auxiliary information. There are many types of deep learning models that are utilized today to enhance the recommendation quality such as Autoencoder (AE), Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Generative Adversarial Network (GAN).

2.3.1 Autoencoder

Deep autoencoders are unsupervised deep learning techniques where the input and output of the network have the same dimension size. They are typically composed of two main parts – the encoder and the decoder – such that the encoder compresses its input into a lower dimensional representation, while the decoder tries to estimate the original input using the lower dimensional representation. By doing this, the autoencoder can be used to reduce input dimensionality while preserving the abstract information of the input.

Autoencoder-based models have been integrated in various recommendation tasks [34, 38, 39, 40, 41, 42, 43]. They are usually used to pretrain data and then use the learned lower representation of that data in the training of another model. Recently, Collaborative Deep Learning (CDL) [34] has become a very popular deep learning technique in RSs due to its promising performance. It is composed from a Stacked Denoising Autoencoder (SDAE) and Probabilistic Matrix Factorization (PMF). CDL can be viewed as an updated version of the Collaborative Topic Regression (CTR) [4] by substituting the Latent Dirichlet Allocation (LDA) topic modeling with a deep-based model, SDAE, to learn from item contents, and then integrating the learned

latent features into PMF. In addition, Deep Collaborative Filtering (DCF) [44] is a similar work that uses a marginalized Denoising Autoencoder (mDA) with PMF. More recently, Collaborative Variational Autoencoder (CVAE) [39] has been used to learn deep item latent features via a variational autoencoder. The authors of CVAE show that their model learns better item features than CDL because their model infers the latent variable distribution in the latent space instead of the observation space.

2.3.2 Multilayer Perceptron

Multilayer Perceptron (MLP) is another deep learning model that is trained in a supervised manner. MLP can be defined as a feedforward neural network with multiple hidden layers between the input and the output layers, where it can be applied to both regression and classification tasks. The structure of MLP can have various shapes like the tower shape used in [45, 46, 47], the constant shape used in [48], or the diamond shape used in [49].

MLP has been recently utilized in several models. For example, the YouTube recommender system in 2016 used a MLP in its model [45]. Specifically, the architecture of the YouTube recommender system had two deep MLPs. The first deep MLP model was used for candidate generation, while the second one was used for ranking. Their evaluation showed that this new deep technique outperformed the YouTube's old recommender system, which implemented using the Matrix Factorization (MF) method. Furthermore, Google presented a combination of wide and deep models for recommender systems [47]. It has shown that the wide model is good for memorization, while the deep model is good for generalization. The deep model uses a MLP with embeddings for sparse features. The model was evaluated on Google

Play and the results showed that both wide and deep model have an improvement in the App acquisition over using wide-only or deep-only models. In addition, Neural Collaborative Filtering (NCF) [46] uses a MLP to model implicit feedback data only, without utilizing users' and items' information into the model training. Later, Joint Neural Collaborative Filtering (J-NCF) [50] was shown to optimize NCF by modeling both feature extraction and user-item interaction via two networks.

2.3.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are very popular in the computer vision domain, especially for grid-like data such as images, due to the huge success they achieve in detecting the spatial relationships in images compared to other traditional models. CNNs typically consist of convolutional layers followed by pooling layers and then a fully connected layer at the end [51]. The convolutional layers extract features from their input by scanning the image using different filter matrices and then generating k feature maps, where k is the total number of filters. The pooling layers are then responsible for reducing the dimensionality of the feature maps. CNNs generally have fewer parameters than MLP using the same number of units, which makes them easy to train [52].

CNNs met recommendations in multiple models [29, 53, 54, 55, 56, 57]. For instance, Convolutional Matrix Factorization (ConvMF) [53] combines CNN with Probabilistic Matrix Factorization (PMF). It effectively utilizes the contextual information of documents in order to extract items' latent factors. The bag-of-word model does not work well since it ignores the word order. Therefore, the authors of ConvMF use CNN to generate the document latent vector, and they then integrate it with the epsilon variable in the PMF model to generate the final prediction. Because ConvMF

only considers an item's auxiliary information, the probabilistic hybrid model (PHD) [54] is proposed, which is built on top of the ConvMF by adding a stacked denoising autoencoder (SDAE) into the model to extract users' latent factors from the users' auxiliary information. By extracting both users' and items' latent factors, the PHD model outperforms ConvMF. Also, Pubmender [55] uses a deep CNN to recommend venues for publishing biomedical articles by using the abstract of each article. Their CNN model is composed of three convolutional and max-pooling layers. Finally, DeepCoNN [56] uses two parallel CNNs to learn from text reviews. They group the reviews by users and items to learn user behaviors and item properties separately in order to predict the corresponding ratings.

2.3.4 Recurrent Neural Network

Recurrent Neural Network (RNN) is another deep learning approach that works very well for sequential data such as text, speech, video, etc. Each unit in the RNN is connected to the previous one, such that it remembers past computations. There are many variants of RNN such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The GRU has less computation cost than LSTM, which makes it more popular.

RNN has been successfully applied in recommendations [3, 58, 59, 60]. For example, AskTheGRU [58] uses a RNN-based technique to represent textual data as latent vectors for the task of scientific article recommendation. Unlike other models that use the bag-of-words for features extraction, the RNN-based model improves the recommendation performance.

2.3.5 Generative Adversarial Network

Generative Adversarial Networks (GANs) are one of the most prevalent recent innovations in the deep learning community. GANs are basically generative models that can generate new data that is similar to the training data. The structure of a GAN has a generative network (a.k.a. the generator) and a discriminative network (a.k.a. the discriminator), such that the generative network produces new data that looks like real data and then the discriminative network decides if a data instance is real (i.e., taken from the actual dataset) or fake (i.e., generated by the generative network). Therefore, the real data works as positive instances for the discriminator, while the fake data works as negative instances. During training, the generator and the discriminator compete against each other, such that the generator's performance improves over training and the discriminator's performance degrades over time and eventually can no longer easily classify between real and fake data.

GANs are very popular applications for image generation, video generation, and voice generation. With that being said, GANs have been effectively employed for recommendation tasks [61, 62, 63, 64]. For example, LARA [64] is a GAN-based model where the generative network generates user profiles corresponding to item attributes, while the discriminative network distinguishes the generated profiles from the real ones. New items are recommended to users who have similar attributes like those generated profiles.

Chapter 3

DeepHCF: Coupling MLP and CNN for Estimating User Ratings

3.1 Abstract

Data sparsity is a significant challenge for collaborative filtering methods in recommendation systems for making accurate recommendations. Several approaches have been published to address this issue. Most of them usually use only one source of data to train their model, and other approaches still have lower performance, especially when the sparsity of data is very high. In this work, we use a deep learning-based model, DeepHCF, to tackle this problem. DeepHCF uses two sources of data (i.e., user-item ratings matrix and item reviews) to train two deep models via joint training. The user-item ratings matrix data is trained using a Multilayer Perceptron (MLP), while other side information is trained using a Convolutional Neural Network (CNN). The users' and items' latent features learned by each model are utilized by factorization machines for our model prediction. Extensive experimental results on four different real-world datasets show that DeepHCF achieves, on average, a 7.42% im-

provement over the second most accurate method, when the dataset has over 99.9% sparsity.

3.2 Introduction

Recommender systems have become an essential component of any commercial website nowadays. These systems are able to gather users' feedback by encouraging users to rate products of their choice, and the systems then use them to improve the recommendation quality. For instance, movie recommendations have gained a lot of attention recently due to the fact that on-demand TV services have played increasing roles in people's lives. According to a research conducted by the Leichtman Research Group in 2018 [65], 69% of United States households subscribe to at least one streaming video service, such as Netflix, Amazon Prime, and Hulu. Moreover, these on-demand TV services are overtaking traditional TV providers. Aside from the absence of the advertisement breaks, people prefer on-demand TV services because of the fact that they can watch TV programs when and where they want. Accordingly, online streaming services integrate RSs to attract more customers, gain customer satisfaction from those using the service, and eventually increase company revenue.

In contrast to the huge amount of research on traditional algorithms for recommender systems like matrix factorization, there is much less work of using deep learning in recommendations. Thus, in this work, we propose DeepHCF, a deep learning model that jointly trains a Multilayer Perceptron (MLP) with a Convolution Neural Network (CNN) for collaborative filtering applications. Some of the current work using deep learning models neglect user-item ratings matrices and focus only on auxiliary information like item reviews [53, 56], while other works show the benefit

of using the ratings matrix to train a MLP [46, 66]. However, using only one source of data usually does not scale well when data sparsity is very high.

Consequently, DeepHCF is trained on two different sources of data using two sub-networks. The two sub-networks are trained jointly. Joint training is different than ensembles, where models in ensembles are trained separately at the training phase [47]. However, models in joint training are trained together at the training phase under the same objective function. The MLP network takes the user-item ratings matrix as an input, while the CNN network takes the other side information as an input. Lastly, prediction layer with factorization machines are introduced on top of the previous sub-networks as the prediction estimator of our model. Figure 3.1 shows the abstract architecture of our model.

DeepHCF is evaluated on four distinct datasets with various sparsity. MovieLens-1M is a relatively denser dataset, with 95% sparsity, compared to Amazon datasets, which have over 99.9% sparsity. DeepHCF outperforms all other methods on Amazon datasets in terms of prediction error. More precisely, DeepHCF achieves a comparable result with the best method on the MovieLens-1M dataset, while it achieves a substantial improvement, 7.42% on average, on Amazon datasets.

The main contributions of this work are summarized as follows:

- We introduce DeepHCF, a deep learning-based model as a hybrid collaborative filtering approach consisting of a Multilayer Perceptron and a Convolutional Neural Network, which takes two different sources of data as input.

- We implement a prediction layer on top of the two deep models, such that user-item and review representations can interact with each other to predict final ratings using factorization machines.

- We evaluate our model on four different real-world datasets with different per-

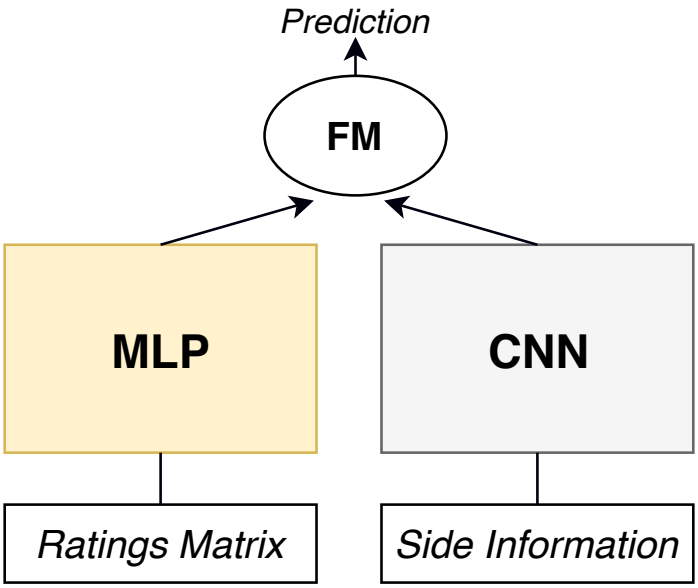


Figure 3.1: DeepHCF overview.

centages of sparsity. We compare the performance of our proposed model with five state-of-the-art approaches. DeepHCF achieves superior performance when the data sparsity is extremely high.

The rest of this chapter is organized as follows. In Section 3.3, we describe the background of this work. Our model, DeepHCF, is described in depth in Section 3.4. Section 3.5 presents the experiments that we have done to evaluate our model against the state-of-the-art approaches. Finally, the conclusions are presented in Section 3.6.

3.3 Background

In this section, we explain the idea behind the factorization machines approach that we use in our model.

3.3.1 Factorization machines

Factorization Machines (FM) [67] is a supervised learning approach that models all possible interactions among feature variables. Learning from features in pairs sometimes has more valuable information than learning from features independently. For example, the two features "toys" and "age" are well correlated to each other such that knowing the user's young age indicates the high possibility that the user would purchase or use a particular toy. Some obvious correlations between any two features can be captured manually by human experts like in the previous example, however, other complex relations between features are hidden in data that needs to be discovered automatically. Thus, FM was originally designed to model all features' interactions. FM captures all single (first-order) and pairwise (second-order) interactions of the input features as follows:

$$FM(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j, \tag{3.1}$$

where w_0 is the global bias, w_i is the weight assigned to each feature, and w_{ij} is the weight assigned for each feature pair $x_i x_j$. w_{ij} can be calculated as the dot product of the embedding vector of feature i (V_i) and the embedding vector of feature j (V_j) as $w_{ij} = V_i^T V_j$. A two-way FM with degree, d=2, is sufficient in most cases, like what is shown in Equation 3.1.

FM has been successfully integrated in recommendation systems [56, 68, 47]. CoNN [56], for example, jointly trains two CNNs that are coupled in the last layer. One CNN is used to train only the user reviews, while the other one is used to train the item reviews. MF is used to predict the corresponding rating at the final layer. One major limitation of this approach is that CoNN is unable to deal with users and item with no previous ratings (a.k.a. the cold start problem).

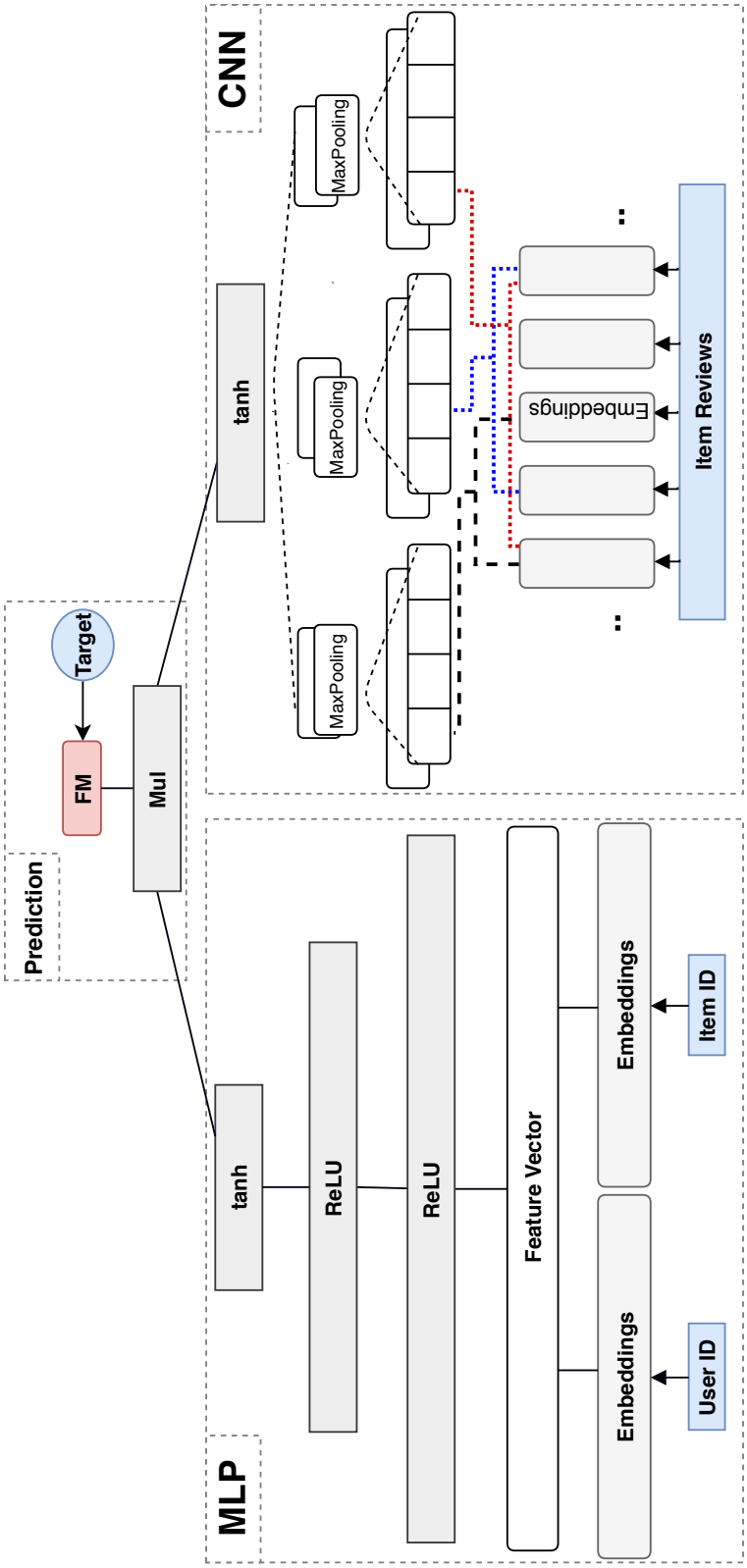


Figure 3.2: DeepHCF architecture.

3.4 Methodology

Our proposed model, DeepHCF, is described thoroughly in this section. DeepHCF is a hybrid model where it takes advantage of both a ratings matrix and other side information to address the sparsity problem. It models users' behavior and items' characteristics using two sources of data: user-item ratings matrix and item reviews. It learns hidden latent features from both sets of data, such that the learned latent features can be employed to approximate ratings for each user. We first describe the general architecture of our model in Section 3.4.1. Then, MLP layers, CNN layers, and prediction layer are presented in Sections 3.4.2, 3.4.3, and 3.4.4, respectively.

3.4.1 Architecture

The architecture of our model for ratings prediction is shown in Figure 3.2. The DeepHCF network consists of two parallel sub-networks, connected by a prediction layer at the top. The two parallel sub-networks comprise a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN) trained in a joint manner to predict final ratings in order to minimize prediction error. MLP is a widely used model with excellent scalability, generalization, performance, and the ability to learn complex relationships. CNN, similarly, is an impressive approach for extracting high-level representations from raw data. The learned latent features of each model are multiplied by each other and then factorized by factorization machines to estimate ratings values. The difference between our predicted value and the actual value is then computed to estimate the loss function of our model. Further specifics about all layers in the architecture are presented in the next three sub-sections.

3.4.2 MLP layers

The MLP structure in our model has three main components: input layer, embedding layer, and hidden layers. User ID and item ID are the inputs of our MLP model, and they are separately embedded into two different embedding layers, which are then concatenated and fed into three hidden layers. Embedding is a function, f: $X \rightarrow \mathbb{R}^n$, that mapping individual data values from a large, sparse categorical domain to dense vectors. It is particularly useful when some of the values may be similar in some meaningful way to other values.

Next, the three hidden layers are in a tower shape, meaning that each hidden layer has fewer neurons than the previous layer. The activation of each neuron $a_j^{(\ell)}$ is computed as:

$$a_j^{(\ell)} = \sigma(\sum_{i=1} a_i^{(\ell-1)} w_{ij}^{(\ell)} + b_j^{(\ell)}), \tag{3.2}$$

where (ℓ) is the layer number and σ is a non-linear activation function. The non-linear activation function used in the first two hidden layers is the Rectified Linear Unit ($ReLU$), which is defined in Equation 3.3, while the third hidden layer has a "$tanh$" activation function (Equation 3.4) to be consistent with the CNN latent feature layer. The learned latent features in the MLP model represents user behavior in collaborative filtering approaches.

$$f(x) = max\{0, x\} \tag{3.3}$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.4}$$

3.4.3 CNN layers

The CNN model in this work is inspired by [69]. The CNN structure has five main components: an input layer, embedding layers, convolution layers, pooling layers, and a fully connected layer at the end. First, distinct words from the item review are embedded into different embedding layers. Word embedding is very influential in natural language processing techniques. It maps each word onto a dense vector, such that similar words have nearly similar vectors. Plotting all the words in the embeddings space would result in placing words with the same meaning or those that are semantically related close to each other in that space. For example, "love" and "like" would eventually be close to each other and the distance (e.g., the Euclidean distance) between these two words' vectors would be very small. However, "camel" and "pen" would have a very large distance between their vectors in the embeddings space because they are semantically different. Glove [70], which is based on matrix factorization techniques, and word2vec [71], which is based on neural networks, are two well-known approaches for word embeddings. Word embeddings can be initialized either with pre-trained word embeddings, or randomly, such that it learns through training. Pre-trained word embeddings are usually preferred and enable the training to converge faster.

Second, the convolutional layers extract features from their inputs. They use multiple filters along with three different word window sizes, w, to capture the surrounding words' interactions. Let $x_i \in \mathbb{R}^k$ represent the k-dimensional word embeddings for the i-th word in the sentence. The whole sentence can be represented as the concatenation of all words embedding as $x_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_n$. Each feature, c_i, extracted from convolution layers is computed as:

$$c_i = \sigma(W \cdot x_{i:i+w-1} + b), \tag{3.5}$$

where b is the bias, $W \in \mathbb{R}^{wk}$ is the filter, and σ is a non-linear activation function. We use ReLU as activation function for the convolutional layers for faster convergence and to reduce the vanishing gradient problem. The feature map, j, generated from convolutions (Equation 3.6) are then fed into the max-pooling layer to capture the most important information.

$$c^j = (c_1^j, c_2^j, c_3^j, c_4^j, ..., c_{n-w+1}^j) \tag{3.6}$$

Max-pooling layer reduces the dimensionality of each feature map by taking the maximum value from values in Equation 3.6. Lastly, the max-pooling layer feeds its output into a fully connected layer with a "$tanh$" activation function. The size of this layer is equivalent to the size of the last hidden layer of the MLP model.

3.4.4 Prediction layer

The outcome of the MLP model and the CNN model are the learned latent features with the same dimensions. The prediction layer concatenates the two sets of latent features and applies a multiplication function (Equation 3.7). The multiplication function is the element-wise multiplication that takes two vectors of the same size, n, and return a single vector of size n also. It simply multiplies vectors A and B, element by element.

$$C^n = (A^n \times B^n) = A_i \times B_i \tag{3.7}$$

The result of the multiplication function is then fed to the factorization machines (FM) layer that applies a sigmoid function (Equation 3.8) to its output. The sigmoid neuron outputs a continuous range of values between 0 and 1 as the predicted rating

of our model.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.8}$$

Therefore, it is essential to normalize our ratings before the training phase to be in the same range [0,1], instead of being in the range of [1,5]. In the testing phase, we need to convert the predicted rating back into the original range of [1,5] by de-normalization. The normalization process is shown in Equation 3.9, while the de-normalization process is shown in Equation 3.10:

$$N(\hat{x_i}) = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3.9}$$

$$D(x_i) = \hat{x_i} \times (max(x) - min(x)) + min(x), \tag{3.10}$$

where $min(x)$ is the minimum rating value, $max(x)$ is the maximum rating value, and i refers to each sample in the dataset.

Since the output of our model is a real-value between 0 and 1, the most straightforward and suitable loss function for the training is the binary cross entropy (Equation 3.11). As it is shown in the equation, the cross entropy loss value increases as the predicted rating value varies from the actual rating value.

$$Entropy(y_i, \hat{y_i}) = -(y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})), \tag{3.11}$$

where y_i is the actual rating value, $\hat{y_i}$ is the predicted rating value, and i refers to each sample in the dataset.

3.5 Experiments

We have performed extensive experiments on different datasets to evaluate our model against other state-of-the-art approaches. Our model is implemented in Python with Tensorflow [72], a well-known Python library for deep learning implemented by Google. We first describe the datasets and evaluation metrics that are used to evaluate DeepHCF in Section 3.5.1. Then, we describe the state-of-the-art approaches in Section 3.5.2. The experimental settings are then defined in Section 3.5.3. Section 3.5.4 shows the impact of using pre-training for word embeddings. Afterwards, the impact of tuning some hyper-parameters on the performance of DeepHCF is reported in Section 3.5.5 Lastly, the performance results are shown in Section 3.5.6.

3.5.1 Datasets

In our experiments, we use four real-world datasets from two different platforms for our evaluation. Three datasets are collected from Amazon product data[1], while the remaining dataset is taken from MovieLens[2]. The four datasets are categorized into two categories based on the platform from which they are collected, which are the following:

- **MovieLens**: This movie rating dataset has been widely used to evaluate collaborative filtering approaches. This version of this dataset has one million ratings with a sparsity of around 95.35%. It is considered a relatively denser dataset compared to Amazon datasets. It has explicit ratings that go from one to five. However, the MovieLens dataset does not have movie reviews in its original format. Therefore, reviews for each movie are obtained from IMDB[3].

[1]http://jmcauley.ucsd.edu/data/amazon
[2]https://grouplens.org/datasets/movielens
[3]http://www.imdb.com

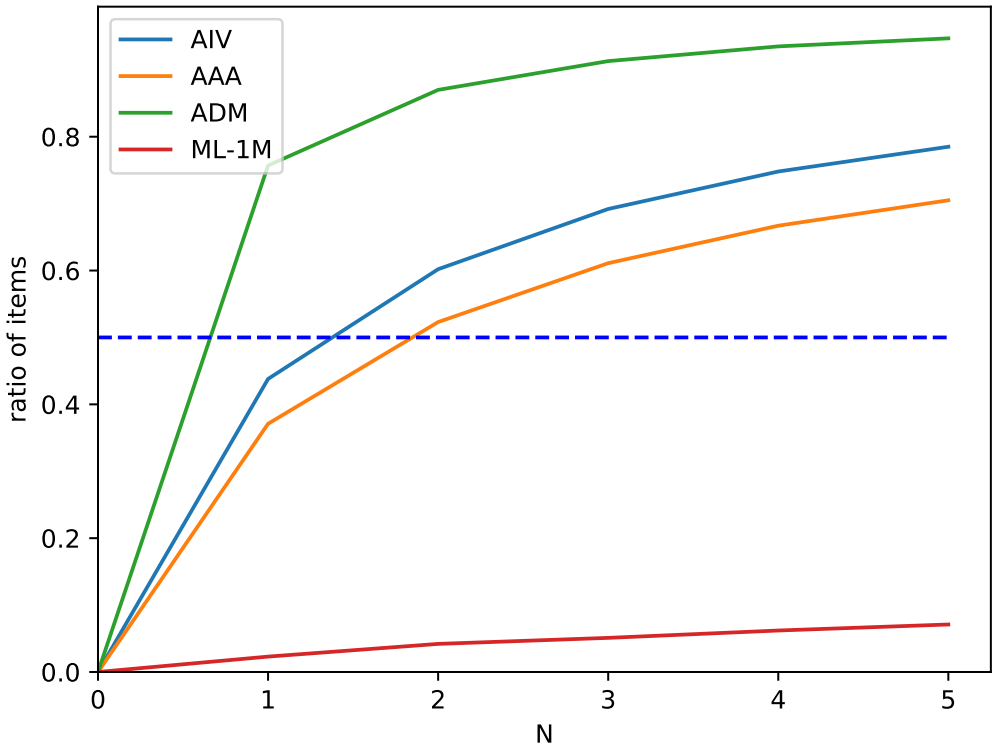


Figure 3.3: Ratio of items that have been rated by N or fewer users in Amazon Instant Video (AIV), Amazon Android Apps (AAA), Amazon Digital Music (ADM), and MovieLens-1M (ML-1M) datasets.

- **Amazon**: This category has three datasets: Amazon Instant Video, Amazon Android Apps, and Amazon Digital Music. This category is critical for most existing approaches due to its high data sparsity that exceeds 99.9%. All datasets contain explicit ratings from one to five and also contain product reviews from Amazon, which are collected between May 1996 and July 2014. More details about the datasets are shown in Table 3.1.

Figure 3.3 shows the ratio of items that have been rated only by five or fewer users. As the plot shows, more than half of items in Amazon datasets have only two or fewer ratings. Moreover, 95% of items in Amazon Digital Music dataset have five

Table 3.1: Description of MovieLens and Amazon datasets.

Dataset	#Users	#Items	#Ratings	Sparsity%
MovieLens-1M	6,040	3,544	993,482	95.35%
Amazon Instant Video	29,757	15,149	135,188	99.97%
Amazon Android Apps	240,931	51,598	1,322,838	99.98%
Amazon Digital Music	56,810	156,493	351,762	99.99%

or fewer ratings. On the contrary, MovieLens-1M is much denser, where at least 90% of movies have been rated by more than five users.

We split each dataset randomly into a training set (80%), a validation set (10%), and a test set (10%). The training set is used to train our model. The validation set is used to stop the model's training early to avoid overfitting. This technique is a form of regularization. The test set is used to compute the performance of our model against the state-of-the-art methods. We use the same sets with all the state-of-the-art methods for the purpose of fair comparison.

Mean Absolute Error (MAE) and Root-Mean-Square Error (RMSE) are used for our evaluations. Lower MAE (Equation 3.12) and lower RMSE (Equation 3.13) indicate a better performance.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.12}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.13}$$

3.5.2 State-of-the-art approaches

Since Convolutional Matrix Factorization (ConvMF+) [53] was extensively evaluated against previous state-of-the-art models like Probabilistic Matrix Factorization

(PMF) [19], Collaborative Topic Regression (CTR) [4], and Collaborative Deep Learning (CDL) [34] and was shown to have better performance by a wide margin, we decide not to repeat those same comparisons in this work and only compare our model with ConvMF+ and the following baseline approaches:

- **User Average**: This approach calculates the average rating for each user in the training set and assigns this value for future prediction. If the user is not found in the training set, the global average will be assigned as a prediction.
- **Item Average**: This approach calculates the average rating for each item in the training set and assigns this value for future prediction. If the item is not found in the training set, the global average will be assigned as a prediction.
- **BPMF**: Bayesian Probabilistic Matrix Factorization [20] has proven to be an improvement over PMF by adding a Bayesian treatment into its model. It is trained using Markov Chain Monte Carlo (MCMC).
- **ALS-WR**: Alternating Least Squares with Weighted Regularization [73] is a matrix factorization method that uses alternating least squares (ALS) with weighted lambda regularization.

ConvMF+ [53] uses both ratings matrix and item reviews similarly as DeepHCF to train a CNN with PMF, while the other baselines use only the rating matrix data for their predictions. Although User Average and Item Average approaches seem very naive, we intentionally add these two basic methods to show that some recent approaches, ConvMF+ for example, fail in some cases, something that is described in detailed at the performance comparison section. For BPMF and ALS-WR, we used the GitHub[4] implementation for both approaches.

[4]https://github.com/chyikwei/recommend

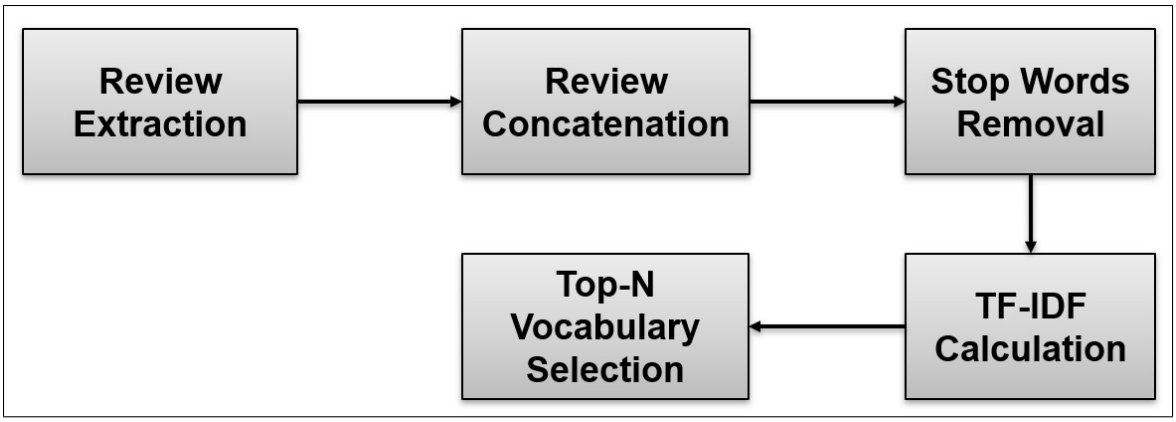


Figure 3.4: A five-stage procedure for preprocessing item reviews.

3.5.3 Experimental settings

We follow the same experimental settings as the aforementioned state-of-the-art approaches [53, 4, 34]. Items that do not have reviews are removed from the dataset. Moreover, users who have fewer than three ratings are also removed in Amazon datasets. After that, the items' reviews in each dataset are preprocessed using a five-stage procedure, as displayed in Figure 3.4. First, each review block has a lot of information (such as "reviewer ID", "item ID", "reviewer name", "rating helpfulness", "review text", "overall rating", "review summary", and "review time"), but we are only interested in two pieces of information from that block: the plain text of the review, and the item ID. An example of what each review block looks like is displayed in Figure 3.5. Second, we group all the items' reviews by the item ID. Third, all stop words (such as "the", "is", or "a") are removed from each review. Stop words are not unique identifiers that can be used to classify and group documents with one another. We also remove all corpus-specific stop words that have a document frequency of more than 0.5. In other words, words that appear in more than 50% of the reviews are removed because they are not considered to be unique identifiers as well. For example, even though words like "movie" and "actor" seem to be unique identifiers

```
{
    "reviewerID": "A21G21TFGZNBUR",
    "asin": "B000GK6NFK",
    "reviewerName": "Angelica Rodriguez Moreno",
    "helpful": [0, 0],
    "reviewText": "It's really good, I totally recommend it for everybody
who likes drama and teenage conflicts. I absolutely love it! :)",
    "overall": 5.0,
    "summary": "Great  TV Show!",
    "unixReviewTime": 1394928000,
    "reviewTime": "03 16, 2014"
}
```


Figure 3.5: An example of one review block from the Amazon Instant Video dataset.

in general, they are not in case of we have reviews from movies dataset; because they appear most frequently in movies' reviews. Fourth, we calculate the *term frequency-inverse document frequency* (TF-IDF) score for each word. TF-IDF [74] is a frequent technique used in the information retrieval field to measure the importance of each word (W) in the collection of documents (D). The TF-IDF score can be calculated as the multiplication of the *term frequency* (TF) by the *inverse document frequency* (IDF), i.e., TF-IDF = TF $\cdot$ IDF, such that:

$$TF = \frac{\text{\# of occurrences of word W in document D}}{\text{Total \# of words in document D}} \tag{3.14}$$

$$IDF = \log\left(\frac{\text{Total \# of documents}}{\text{\# of documents containing word W}}\right) \tag{3.15}$$

where $IDF = 0$ if a word appears in all documents. Stop words usually have this value. Therefore, this metric is very practical for filtering words, such that frequent words have lower scores while unique words have higher scores.

Finally, after the TF-IDF score of each word is calculated, only the top 8000 distinct words are selected. Also, we limit the maximum review length to 200 words.

For MLP layers, we set the embedding size for both user and item attributes to 64 dimensions. Also, we use the dropout technique with values of [0.1 and 0.25] for

regularization. In the same way for CNN, we use three different window sizes [1, 2, and 3] for the convolutional layers to represent the surrounding words, and we use 64 filters per window size. In addition, we use a dropout rate equal to 0.25 to avoid overfitting. Lastly, the pre-trained word embeddings[5] of 200 dimensions are used to initialize the word embedding vectors.

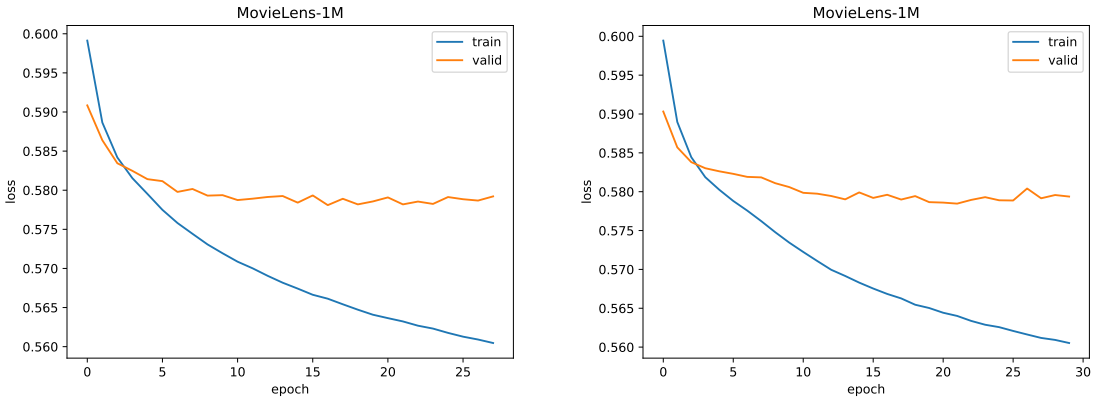
3.5.4 Impact of pre-training

Figures 3.6 and 3.7 show the state of each epoch of our model, DeepHCF, with and without pre-training CNN embeddings on MovieLens-1M and Amazon Instant Video dataset. We can observe from Figures 3.6 and 3.7 that the Amazon datasets in general converge very quickly with only two epochs; this is due clearly to the high sparsity of the datasets, while MovieLens-1M converges with 18 epochs of training. On the other hand, with or without Glove word embeddings to initialize our CNN embeddings vectors as pre-training, DeepHCF displays a fast convergence within both datasets. Thus, pre-training does not improve our model performance, and a random initialization can achieve results equivalent to the results with word embeddings pre-training. In contrast to the significant impact of pre-training on other approaches, this result demonstrates that DeepHCF is insensitive to parameter initialization of the embedding layer.

3.5.5 Impact of hyper-parameters tuning

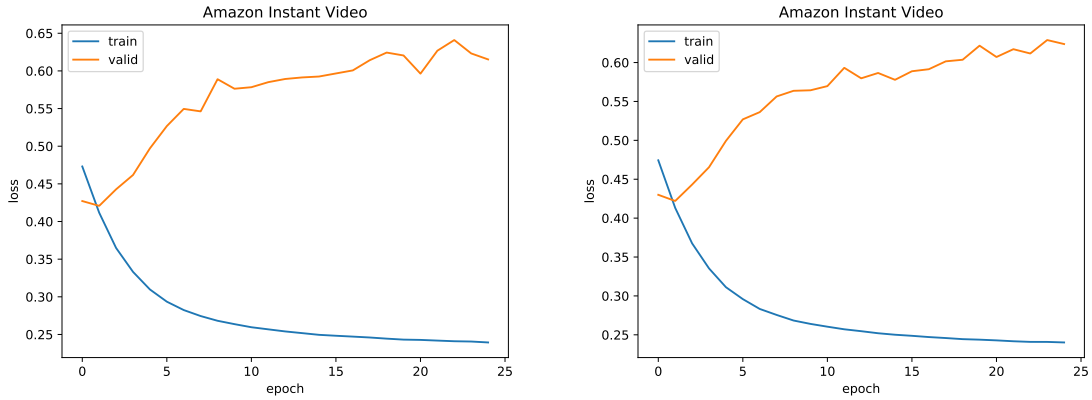
To determine the best hyper-parameters values that maximize our model performance, we run our model on different setting values regarding the dimension size of the MLP embeddings and the batch size of the MovieLens-1M (MovieLens) and

[5]http://nlp.stanford.edu/projects/glove



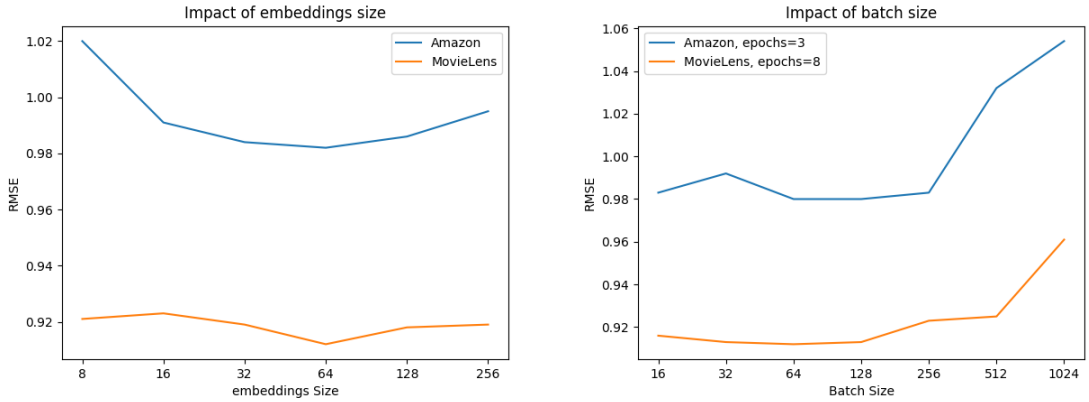
(a) With pre-training (b) Without pre-training

Figure 3.6: Training and validation loss values of each epoch using the MovieLens-1M dataset, with and without pre-training.



(a) With pre-training (b) Without pre-training

Figure 3.7: Training and validation loss values of each epoch using the Amazon Instant Video dataset, with and without pre-training.



(a) Embeddings size

(b) Batch size

Figure 3.8: The impact of hyper-parameters tuning on DeepHCF performance for: (a) dimension of MLP embeddings, and (b) batch size.

the Amazon Instant Video (Amazon) datasets. First, when we set the embeddings dimension for our MLP network to one of the following values {8, 16, 32, 64, 128, 256}, we find that the best performance occurs when we set the dimension to 64, as Figure 3.8a shows. Similarly, we repeat the same procedure for the batch size, where values are in the following range {16, 32, 64, 128, 256, 512, 1024}. We find that using more than 256 training examples decrease the performance, and the optimal batch size in our case is between 128 and 256, as Figure 3.8b shows.

3.5.6 Performance comparison

For each dataset, we run three experiments, with the average error is reported in this section. The performance of DeepHCF and the other baselines are reported in the next two sub-sections. DeepHCF outperforms all baselines on the Amazon datasets and shows competitive results on the MovieLens-1M dataset according to both metrics: MAE and RMSE. The improvement rate in each table indicates the improvement in error for our model, DeepHCF, over the second most accurate method.

Table 3.2: Performance results of our model, DeepHCF, against other baselines using the MovieLens-1M dataset. Best values are marked in bold.

Approach	MovieLens-1M	
	MAE	RMSE
User Average	0.830	1.036
Item Average	0.781	0.979
BPMF	0.678	0.869
ALS-WR	0.675	**0.861**
ConvMF+	0.678	**0.861**
DeepHCF	**0.674**	0.867
Improvement%	0.14%	-

Table 3.3: Performance results of our model, DeepHCF, against other baselines using Amazon Instant Video (AIV), Amazon Android Apps (AAA), and Amazon Digital Music (ADM) datasets. Best values are marked in bold.

Approach	AIV		AAA		ADM	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
User Average	0.778	1.168	1.005	1.373	0.569	0.917
Item Average	0.788	1.079	0.975	1.250	0.625	0.943
BPMF	0.817	1.103	1.024	1.326	0.612	0.910
ALS-WR	0.856	1.141	1.087	1.423	0.631	0.919
ConvMF+	0.866	1.149	0.994	1.307	0.661	0.934
DeepHCF	**0.688**	**0.994**	**0.896**	**1.180**	**0.520**	**0.830**
Improvement%	11.43%	7.87%	8.10%	5.60%	8.61%	8.97%
Average Improvement%	MAE= 9.38%, RMSE= 7.42%					

ConvMF+ has two attributes, λ_i and λ_j, that should be manually set and are significant to its performance. Thus, we use the same values the original authors used in their paper for the MovieLens-1M, i.e., $\lambda_i = 100$ and $\lambda_j = 10$. However, for all Amazon datasets, we search in the grid of [1, 10, 100] and find that best performance is given when $\lambda_i = 1$ and $\lambda_j = 100$.

MovieLens-1M dataset

This dataset has a level of sparsity equal to 95.35%. Table 3.2 summarizes the performance on this dataset. As the table shows, BPMF, ALS-WR, and ConvMF+ perform very well and have much better performance than the other straightforward methods, User Average and Item Average. DeepHCF as well performs well on this dataset and achieves a comparable performance with other baselines. This result demonstrates that most of the existing collaborative filtering approaches, including DeepHCF, are effective and powerful with relatively denser datasets.

Amazon datasets

This category has three datasets, all of which are extremely sparse, something that poses a significant challenge to all collaborative filtering approaches to make accurate recommendations. All datasets have a sparsity of more than 99.9%. Table 3.3 summarizes the performance of all methods on Amazon datasets. We can observe from Table 3.3 that ConvMF+ performs poorly compared to other approaches in this category due to the high percentage of sparsity. Furthermore, the two naive methods, Item Average and User Average, have a better score than ConvMF+ in terms of both metrics. We believe that the user latent vector of ConvMF+, which is learned by PMF, does not represent the user features very accurately in the Amazon datasets.

On the contrary, our model, DeepHCF, achieves a substantial improvement over the second most accurate method in Amazon Instant Video, Amazon Android Apps, Amazon Digital Music datasets by a marginal improvement in RMSE of 7.87%, 5.60%, and 8.97% respectively. The average improvement in RMSE over the three datasets is 7.42%. This significant gain demonstrates that the DeepHCF model works much better than the other methods on very sparse datasets, such as the Amazon datasets. We believe that using MLP to learn the user representation, and CNN to analyze the sentiment in text reviews are more powerful and robust with extraordinarily sparse datasets.

3.6 Conclusion

In this work, we develop a deep learning method to address the data sparsity problem of collaborative filtering approaches in recommendation systems. We introduce DeepHCF, a deep learning network for hybrid collaborative filtering, composed of two sub-networks trained jointly with two different sources of data. A prediction layer with factorization machines is built on top of the two models to utilize their outcome for our final prediction. We evaluate our model on four different real-world datasets against five methods. The evaluation results show that DeepHCF achieves superior performance in both metrics, MAE and RMSE, in the case of three datasets with extreme data sparsity, while it achieves a competitive performance when the data is relatively denser. In conclusion, our model is effective on both dense and sparse data.

Chapter 4

CATA: A Collaborative Attentive Autoencoder Method for Recommending Scientific Articles

4.1 Abstract

Matrix Factorization (MF) is a successful collaborative filtering approach used in recommendation systems. However, its performance decreases significantly when users of the system have limited, inadequate feedback data. This problem is also known as the data sparsity problem. To handle this problem, hybrid approaches were proposed recently to integrate items' contextual information with MF-based approaches, which improved the performance of recommendations. Nevertheless, learning better representation of the items' contents is still a challenge that needs to be further enhanced. In this work, we propose a Collaborative Attentive Autoencoder (CATA) that learns latent factors of items through an attention mechanism that can capture the most pertinent part of information for making better recommendations. Comprehensive

experiments on two real-world datasets have shown our method performs better than the state-of-the-art models according to various evaluation metrics.

4.2 Introduction

Scientific article recommendation is a very common application for RSs. It keeps researchers updated on recent related work in their field. One traditional way to find relevant articles is to go through the references section in other articles. Yet, this approach is biased toward heavily cited articles, such that new relevant articles with higher impact have less chance to be found. Another method is to search for articles using keywords. Although this technique is popular among researchers, they must filter out a tremendous number of articles from the search results to retrieve the most suitable articles. Moreover, all users get the same search results with the same keywords, and these results are not personalized based on the users' personal interests. Thus, recommendation systems can address this issue and help scientists and researchers find valuable articles while being aware of recent related work.

Deep learning (DL) recently has become an effective approach for most computer science problems. DL meets recommendation systems the last few years and has shown superiority over traditional CF models. One of the first works to apply DL for CF recommendations is Restricted Boltzmann Machines (RBM) [37]. However, this approach is not deep enough (two layers only) to learn users' tastes from their histories, and it does not take contextual information in consideration. Later on, Collaborative Deep Learning (CDL) [34] became the state-of-the-art method in DL-based RSs. Recently, Collaborative Variational Autoencoder (CVAE) [39] has been proposed which uses a variational autoencoder to handle the item contents, and has shown to have better predictions over CDL. However, both CDL and CVAE mod-

els assume that all parts of their model's contribution are the same for their final predictions.

Hence, in this work, we propose a deep learning-based model named Collaborative Attentive Autoencoder (CATA) for recommending scientific articles. In particular, we integrate the attention mechanism into our unsupervised deep learning process to identify an item's features. We learn the item's features from the article's textual information (e.g., the article's title and abstract) to enhance the recommendation quality. The compressed low-dimensional representation learned by the unsupervised model is incorporated then into the matrix factorization approach for our ultimate recommendation. To demonstrate the capability of our proposed model to generate more relevant recommendations, we conduct inclusive experiments on two real-world datasets, which are taken from the CiteULike[1] website, to evaluate CATA against multiple recent works. The experimental results prove that our model can extract more constructive information from an article's contextual data than other models. More importantly, CATA performs very well where the data sparsity is extremely high.

The remainder of this chapter is organized in the following manner. First, we demonstrate essential background and related work in Section 4.3. Our model, CATA, is introduced in Section 4.4. The experimental results of our model against the state-of-the-art models are discussed thoroughly in Section 4.5. We then conclude our work in Section 4.6.

[1] www.citeulike.org

Table 4.1: A summary description of notations used in this chapter.

Notation	Meaning
n	Number of users
m	Number of articles
d	Dimension of the latent factors
R	User-article matrix
U	Latent factors of users
V	Latent factors of articles
C	Confidence matrix
P	Users preferences matrix
X_j	Article's side information, i.e., title and abstract
T_j	Article's side information, i.e., tags and citations
$\theta(X_j)$	Mapping function for input X_j of the first autoencoder
$\gamma(T_j)$	Mapping function for input T_j of the second autoencoder
Z_j	Compressed representation of X_j
Y_j	Compressed representation of T_j
λ_u	Regularization parameter for users
λ_v	Regularization parameter for articles

4.3 Background

Our work is designed and evaluated on recommendations with implicit feedback. Thus, in this section, we describe the well-known collaborative filtering approach, Matrix Factorization, for implicit feedback problems. After that, we describe one of the recent ideas in deep learning community that is incorporated into our model in this chapter, the attention mechanism. Table 4.1 summarizes all the notations used in chapter 4 and chapter 5 to describe our approaches.

4.3.1 Matrix factorization

Matrix Factorization (MF) [75] is the most popular CF method, mainly due to its simplicity and efficiency. The idea behind MF is to decompose the user-item matrix, $R \in \mathbb{R}^{n \times m}$, into two lower-dimensional matrices, $U \in \mathbb{R}^{n \times d}$ and $V \in \mathbb{R}^{m \times d}$, such that

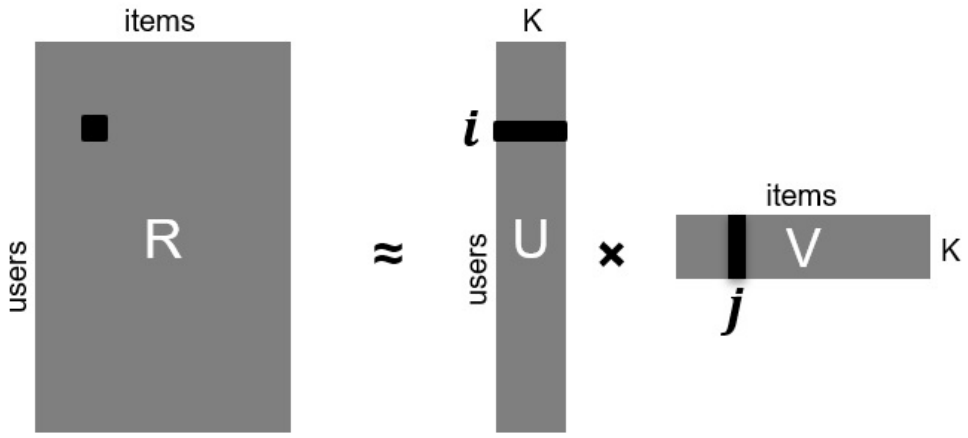


Figure 4.1: Matrix Factorization illustration.

the inner product of U and V will approximate the original matrix R, where d is the dimension of the latent factors, such that $d \ll min(n, m)$.

$$R \approx U \cdot V^T \tag{4.1}$$

Figure 4.1 illustrates the MF process. For example, the approximation rating for user i to item j, r_{ij}, can be computed as the dot product of vector u_i with vector v_j^T.

MF optimizes the values of U and V by minimizing the sum of the squared difference between the actual values and the model predictions with adding two regularization terms, as shown here:

$$\mathcal{L} = \sum_{i,j \in R} \frac{I_{ij}}{2}(r_{ij} - u_i v_j^T)^2 + \frac{\lambda_u}{2} \|u_i\|^2 + \frac{\lambda_v}{2} \|v_j\|^2 \tag{4.2}$$

where I_{ij} is an indicator function that equals 1 if user_i has rated item_j, and 0 if otherwise. Also, $||U||$ and $||V||$ are the Euclidean norms, and λ_u, λ_v are two regularization terms preventing the values of U and V from being too large. This avoid model overfitting.

Explicit data, such as ratings (r_{ij}) are not regularly available. Therefore, Weighted Regularized Matrix Factorization (WRMF) [12] introduces two modifications to the previous objective function to make it work for implicit feedback. The optimization process in this case runs through all user-item pairs with different confidence levels assigned to each pair, as in the following:

$$\mathcal{L} = \sum_{i,j \in R} \frac{c_{ij}}{2}(p_{ij} - u_i v_j^T)^2 + \frac{\lambda_u}{2} \|u_i\|^2 + \frac{\lambda_v}{2} \|v_j\|^2 \tag{4.3}$$

where p_{ij} is the user preference score with a value of 1 when user$_i$ and item$_j$ have an interaction, and 0 otherwise. c_{ij} is a confidence variable where its value shows how confident the user like the item. In general, $c_{ij} = a$ when $p_{ij} = 1$, and $c_{ij} = b$ when $p_{ij} = 0$, such that $a > b > 0$.

Stochastic Gradient Decent (SGD) [76] and Alternating Least Squares (ALS) [73] are two optimization methods that can be used to minimize the objective function of MF in Equation 4.2. The first method, SGD, loops over each single training sample and then computes the prediction error as:

$$e_{ij} = r_{ij} - u_i v_j^T \tag{4.4}$$

The gradient of the objective function with respect to u_i and v_j can be computed as the following:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial u_i} &= -\sum_j I_{ij}(r_{ij} - u_i v_j^T)v_j + \lambda_u u_i \\ \frac{\partial \mathcal{L}}{\partial v_j} &= -\sum_i I_{ij}(r_{ij} - u_i v_j^T)u_i + \lambda_v v_j \end{aligned} \tag{4.5}$$

After calculating the gradient, SGD updates the user and item latent factors in

the opposite direction of the gradient using the following equations:

$$\begin{aligned} u_i &\leftarrow u_i + \alpha(\sum_j I_{ij} e_{ij} v_j - \lambda_u u_i) \\ v_j &\leftarrow v_j + \alpha(\sum_i I_{ij} e_{ij} u_i - \lambda_j v_j) \end{aligned} \tag{4.6}$$

where α is the learning rate. Learning rate is usually set with a very small number (e.g., 0.001). Setting the learning step with too large number may lead into missing the minimum. However, if it is too small, it takes more time for the model to converge.

Even though SGD is easy to implement and generally faster than ALS in some cases, it is not suitable to use with implicit feedback, since looping over each single training sample is not practical. ALS works better in this case. ALS iteratively optimizes U while V is fixed, and vice versa. This optimization process is repeated until the model converges.

To determine what user and item vector values minimize the objective function for implicit data (Equation 4.3), we first take the derivative of $\mathcal{L}$ with respect to u_i.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial u_i} &= -\sum_j c_{ij}(p_{ij} - u_i v_j^T) v_j + \lambda_u u_i \\ 0 &= -C_i(P_i - u_i V^T)V + \lambda_u u_i \\ 0 &= -C_i V P_i + C_i V u_i V^T + \lambda_u u_i \\ VC_iP_i &= u_i V C_i V^T + \lambda_u u_i \\ VC_iP_i &= u_i(VC_iV^T + \lambda_u I) \\ u_i &= VC_iP_i(VC_iV^T + \lambda_u I)^{-1} \\ u_i &= (VC_iV^T + \lambda_u I)^{-1} VC_iP_i \end{aligned} \tag{4.7}$$

where I is the identity matrix.

Similarly, taking the derivative of $\mathcal{L}$ with respect to v_j leads to:

$$v_j = (UC_jU^T + \lambda_v I)^{-1} UC_jP_j \tag{4.8}$$

4.3.2 Attention mechanism

The idea of the attention mechanism is motivated by the human vision system and how our eyes pay attention and focus on a specific part of an image, or specific words in a sentence, for example. In the same way, attention in deep learning can be described simply as a vector of weights to show the importance of the input elements. Thus, the intuition behind attention is that not all parts of the input are equally significant, i.e., only few parts are significant for the model. Attention was initially designed for a neural machine translation system [77], and then successfully applied in other contexts such as image classification [78], and document classification [79].

Recently, attention mechanism has been adopted frequently in natural language processing (NLP) applications, where the it can pay attention to different parts of the text input. In particular, attention is able to determine the contribution of each text's segment by computing the weighted sum of all input segments, and then assigning different scores to each segment via alignment score function. Currently, there are several variations of the attention mechanism, which has been introduced in the literature, such as the global attention [80], the local attention [80], and the self-attention [81]. For instance, the global attention works on the entire input sequence, while the local attention is only applied to a subset of the sequence, such that the local attention has less computational complexity.

Attention has also been successfully applied in different recommendation tasks [82, 83, 84, 85, 86, 87]. For example, MPCN [84] is a multi-pointer co-attention network

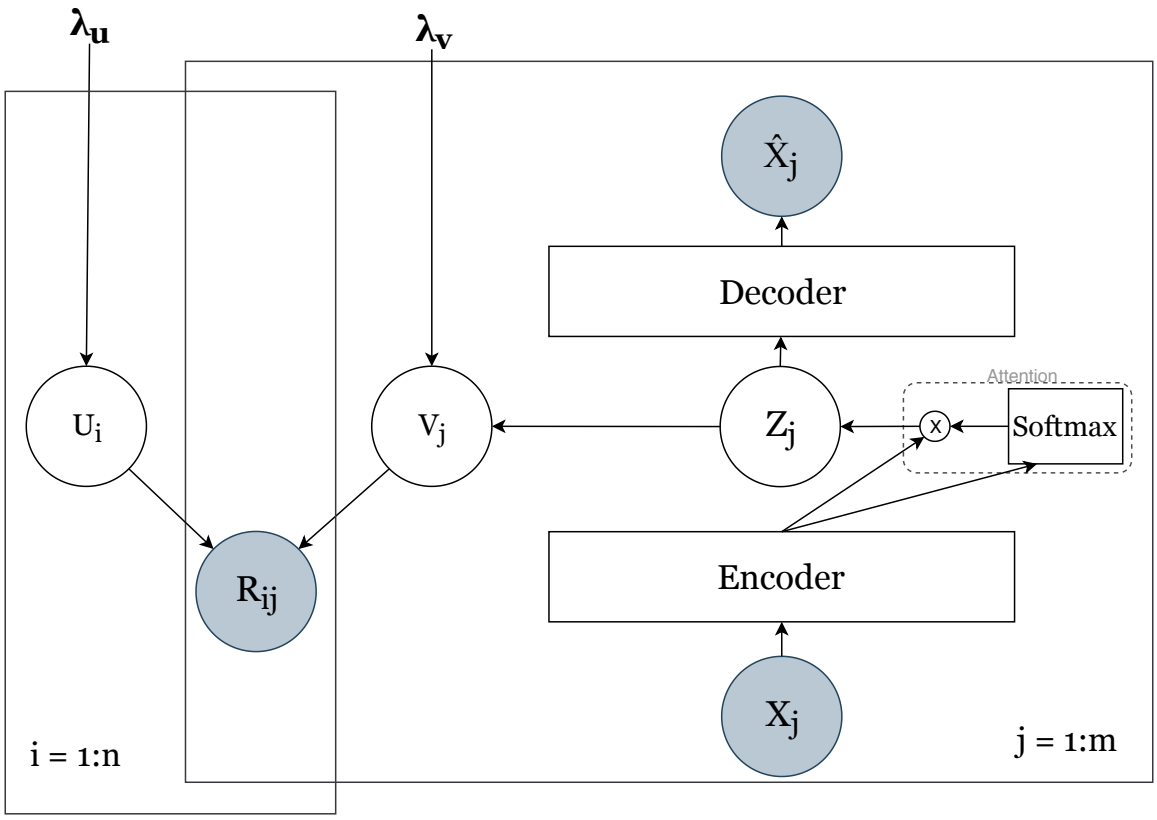


Figure 4.2: Collaborative Attentive Autoencoder (CATA) architecture.

that takes user and item reviews as input, and then extracts the most informative reviews that contribute more in predictions. Also, D-Attn [86] uses a convolutional neural network with dual attention (local and global attention) to represent the user and the item latent representations similarly like the matrix factorization approach. Moreover, NAIS [87] employs an attention network to distinguish items in a user profile, which have more influential effects in the model predictions.

4.4 Proposed model

In this section, we illustrate our proposed model in depth. The intuition behind our model is to learn the latent factors of items in PMF with the use of available side

textual contents via an attentive unsupervised learning that can catch more plentiful information from the available data. The architecture of our model is displayed in Figure 4.2. We clarify each part of our model individually in the next two sections.

4.4.1 The attentive autoencoder

Autoencoder [88] is an unsupervised learning neural network that is useful for compressing high-dimensional input data into a lower-dimensional representation while preserving the abstract nature of the data. The autoencoder network is generally composed of two main components, i.e., the encoder and the decoder. The encoder takes the input and encodes it through multiple hidden layers and then generates a compressed representative vector, Z_j. The encoding function can be formulated as $Z_j = f(X_j)$. Subsequently, the decoder can be used then to reconstruct and estimate the original input, $\hat{X}_j$, using the representative vector, Z_j. The decoder function can be formulated as $\hat{X}_j = f(Z_j)$. Each the encoder and the decoder usually consist of the same number of hidden layers and neurons. The output of each hidden layer is computed as follows:

$$h^{(\ell)} = \sigma(h^{(\ell-1)}W^{(\ell)} + b^{(\ell)}) \tag{4.9}$$

where (ℓ) is the layer number, W is the weights matrix, b is the bias vector, and σ is a non-linear activation function. We use the Rectified Linear Unit (ReLU) as the activation function.

Our model takes input from the article's textual data, $X_j = \{x^1, x^2, ..., x^s\}$, where x^i is a value between $[0, 1]$ and s represents the vocabulary size of the articles' titles and abstracts. In other words, the input of our autoencoder network is a normalized bag-of-words histograms of filtered vocabularies of the articles' titles and abstracts.

Batch normalization (BN) [89] has been proven to be a proper solution for the

internal covariant shift problem, where the layer's input distribution in deep neural networks changes across the time of training, and causes difficulty in training the model. In addition, BN can work as a regularization procedure like Dropout [90] in deep neural networks. Accordingly, we apply a batch normalization layer after each hidden layer in our autoencoder to obtain a stable distribution from each layer's output.

Furthermore, we use the idea of attention mechanism to work between the encoder and the decoder such that only relevant parts of the encoder output are selected for the input reconstruction. It calculates the scores as the probability distribution of the encoder's output using the $softmax(.)$ function.

$$f(z_c) = \frac{e^{z_c}}{\sum_d e^{z_d}} \tag{4.10}$$

The probability distribution and the encoder output are then multiplied using element-wise multiplication function to get Z_j.

We use the attentive autoencoder to pretrain the items contextual information and then integrate the compressed representation, Z_j, in computing the items latent factors, V_j, from the matrix factorization method. The dimension space of Z_j and V_j are set to be equal to each other. Finally, we adopt the binary cross-entropy (Equation 4.11) as the loss function we want to minimize of our attentive autoencoder model.

$$\mathcal{L} = -\sum_k \big(y_k \log(p_k) - (1 - y_k) \log(1 - p_k)\big) \tag{4.11}$$

where y_k corresponds to the correct labels, and p_k corresponds to the predicted values.

The value of p that minimizes the previous loss function the most is when $p = y$, which makes it fit for our autoencoder. To verify that, taking the derivative of the

loss function to respect to p results into:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial p} &= -y(\frac{1}{p}) - (1-y)(\frac{-1}{1-p}) \\
&\frac{-y}{p} + \frac{1-y}{1-p} = 0 \\
&-y(1-p) + (1-y)p = 0 \\
&-y + yp + p - yp = 0 \\
&-y + p = 0 \\
&p = y
\end{aligned} \tag{4.12}$$

4.4.2 Probabilistic matrix factorization

Probabilistic Matrix Factorization (PMF) [19] is a probabilistic linear model where the prior distributions of the latent factors and users' preferences are drawn from Gaussian normal distribution.

$$\begin{aligned}
u_i &\sim \mathcal{N}(0, \lambda_u^{-1} I) \\
v_j &\sim \mathcal{N}(0, \lambda_v^{-1} I) \\
p_{ij} &\sim \mathcal{N}(u_i v_j^T, \sigma^2)
\end{aligned} \tag{4.13}$$

We integrate the items' contents, trained through the attentive autoencoder, into PMF. Therefore, the objective function in Equation 4.3 has been changed slightly to become:

$$\mathcal{L} = \sum_{i,j \in R} \frac{c_{ij}}{2}(p_{ij} - u_i v_j^T)^2 + \frac{\lambda_u}{2} \|u_i\|^2 + \frac{\lambda_v}{2} \|v_j - \theta(X_j)\|^2 \tag{4.14}$$

where $\theta(X_j) = Encoder(X_j) = Z_j$.

Thus, taking the partial derivative of our previous objective function with respect to both u_i and v_j results in the following equations that minimize our objective

function the most:

$$
\begin{aligned}
u_i &= (VC_iV^T + \lambda_u I)^{-1} VC_iP_i \\
v_j &= (UC_jU^T + \lambda_v I)^{-1} UC_jP_j + \lambda_v \theta(X_j)
\end{aligned}
\tag{4.15}
$$

4.4.3 Prediction

After our model has been trained and the latent factors of users and articles, U and V, are identified, we calculate our model's prediction scores of user$_i$ and each article as the dot product of vector u_i with all vectors in V as $scores_i = u_iV^T$. Then, we sort all articles based on our model predication scores in descending order, and then recommend the top-K articles for that user$_i$. We go through all users in U in our evaluation and report the average performance among all users. The overall process of our approach is illustrated in Algorithm 1.

Algorithm 1: CATA algorithm

```
pre-train autoencoder with input X;
Z ← θ(X);
U, V ← Initialize with random values;
while <NOT converge> do
    for <each user i> do
        u_i ← update using Equation 4.15;
    end for
    for <each article j> do
        v_i ← update using Equation 4.15;
    end for
end while
for <each user i> do
    scores_i ← u_i V^T;
    sort(scores_i) in descending order;
end for
Evaluate the top-K recommendations;
```

Table 4.2: Description of CiteULike datasets.

Dataset	#Users	#Articles	#Pairs	Sparsity%
Citeulike-a	5,551	16,980	204,986	99.78%
Citeulike-t	7,947	25,975	134,860	99.93%

4.5 Experiments

In this section, we conduct extensive experiments aiming to answer the following research questions:

- **RQ1**: How does our proposed model, CATA, perform against state-of-the-art methods?
- **RQ2**: Does adding the attention mechanism actually improve our model performance?
- **RQ3**: How could different values of the regularization parameters (λ_u and λ_v) affect CATA performance?
- **RQ4**: What is the impact of different dimension values of users and items' latent space on CATA performance?
- **RQ5**: How many training epochs are sufficient for pretraining our autoencoder?

We first describe the datasets, evaluation methodology, baselines, and then followed by the experimental results answering the previous research questions.

4.5.1 Datasets

Two scientific article datasets are used to evaluate our model against state-of-the-art methods. Both datasets are collected from CiteULike domain. The first dataset

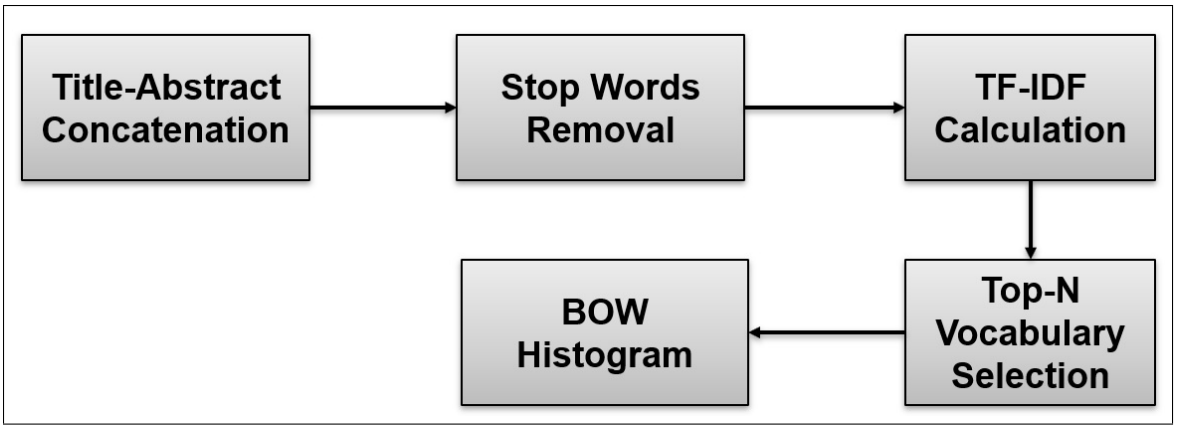
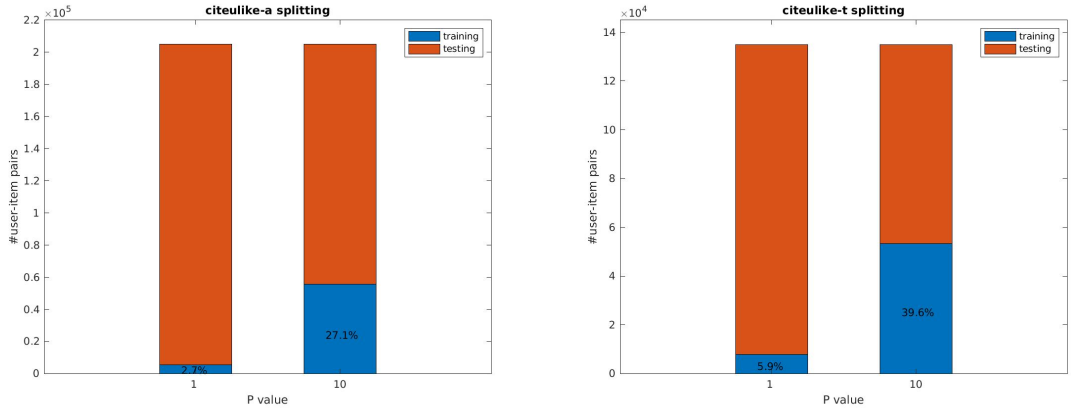


Figure 4.3: A five-stage procedure for preprocessing article titles and abstracts.

is called Citeulike-a, which is collected by [4], has 5,551 users, 16,980 articles, and 204,986 user-article pairs. The sparseness of this dataset is extremely high, where only around 0.22% of the user-article matrix has interactions. Each user has at least ten articles in his or her library. On average, each user has 37 articles in his or her library and each article has been added to 12 users' libraries. On the other hand, the second dataset is called Citeulike-t, which is collected by [32]. It has 7,947 users, 25,975 articles, and 134,860 user-article pairs. This dataset is actually sparser than the first one with only 0.07% available user-article interactions. Each user has at least three articles in his or her library. On average, each user has 17 articles in his or her library and each article has been added to five users' libraries. Brief statistics of the datasets are shown in Table 4.2.

Title and abstract of each article are given. The average number of words per article in both title and abstract after our text preprocessing is 67 words in Citeulike-a, and 19 words in Citeulike-t. We follow the same preprocessing techniques as the state-of-the-art models [34, 4, 39]. A five-stage procedure to preprocess the textual content is displayed in Figure 4.3. Each articles title and abstract are combined together and then are preprocessed such that stop words are removed. After that, top-N distinct



(a) Citeulike-a (b) Citeulike-t

Figure 4.4: The percentage of the data entries that forms the training and testing sets in CiteULike datasets.

words based on the TF-IDF measurement are picked out. 8,000 distinct words are selected for Citeulike-a, while 20,000 distinct words are selected for Citeulike-t to form the bag-of-words histogram, which are then normalized into values between 0 and 1 based on the vocabularies' occurrences.

4.5.2 Evaluation methodology

We follow the state-of-the-art techniques [39, 34, 32] to generate our training and testing sets. For each dataset, we create two versions of the dataset for sparse and dense settings. In total, four dataset cases are used in our evaluation. To form the sparse ($P = 1$) and the dense ($P = 10$) datasets, P items are randomly selected from each user library to generate the training set while the remaining items from each user library are used to generate the testing set. As a result, when $P = 1$, only 2.7% and 5.9% of the data entries are used to generate the training set in Citeulike-a and Citeulike-t, respectively. Similarly, 27.1% and 39.6% of the data entries are used to generate the training set when $P = 10$ as Figure 4.4 shows.

We use recall and Discounted Cumulative Gain (DCG) as our evaluation metrics to test how our model performs. Recall is usually used to evaluate recommender systems with implicit feedback. However, precision is not favorable to use with implicit feedback because the zero value in the user-article interaction matrix has two meanings: either the user is not interested in the article, or the user is not aware of the existence of this article. Therefore, using the precision metric only assumes that for each zero value the user is not interested in the article, which is not the case.

Recall per user can be measured using the following formula:

$$recall@K = \frac{\text{Relevant Articles} \cap \text{K Recommended Articles}}{\text{Relevant Articles}} \tag{4.16}$$

where K is set manually in the experiment and represents the top K articles of each user. We set K = 50, 100, 150, 200, 250, and 300 in our evaluations. The overall recall can be calculated as the average recall among all users. If K equals the number of articles in the dataset, recall will have a value of 1.

Recall, however, does not take into account the ranking of articles within the top-K recommendations, as long as they are in the top-K list. However, DCG does. DCG shows the capability of the recommendation engine to recommend articles at the top of the ranking list. Articles in higher ranked K positions have more value than others. The DCG among all users can be measured using the following equation:

$$DCG@K = \frac{1}{|U|} \sum_{u=1}^{|U|} \sum_{i=1}^{K} \frac{rel(i)}{log_2(i+1)} \tag{4.17}$$

where $|U|$ is the total number of users, i is the rank of the top-K articles recommended by the model, and $rel(i)$ is an indicator function that outputs 1 if the article at rank i is a relevant article, and 0 otherwise.

4.5.3 State-of-the-art approaches

We evaluate our approach against two recent hybrid approaches, as they are described below:

- **CML+F**: Collaborative Metric Learning (CML) [91] is a metric learning model that pulls items liked by a user closer to that user. Recommendations are then generated based on the K-Nearest Neighbor of each user. CML+F additionally uses a neural network with two fully connected layers to train items' features (articles' tags in this work) to update items' location. CML+F has been shown to have a better performance than CML.

- **CVAE**: Collaborative Variational Autoencoder (CVAE) [39] is a probabilistic model that jointly models both user-item matrix and items content using a variational autoencoder (VAE) with a probabilistic matrix factorization (PMF). It can be considered as the baseline of our proposed approach since they share the same strategy.

We omit comparisons with some well-known hybrid collaborative filtering approaches such as CTR [4], CDL [34], DeepMusic [29], and VBPR [92] because they have been already outperformed by CVAE and CML+F.

For hyper-parameter settings, we set the confidence variables (i.e., a and b) to $a = 1$, and $b = 0.01$. These are the same values used in CVAE as well. Also, a four-layer network is used to construct our attentive autoencoder. The four-layer network has the following shape "#Vocabularies-400-200-100-50-100-200-400-#Vocabularies".

Table 4.3: Parameter settings for λ_u and λ_v for our model, CATA, and CVAE.

Approach	Citeulike-a				Citeulike-t			
	Sparse		Dense		Sparse		Dense	
	λ_u	λ_v	λ_u	λ_v	λ_u	λ_v	λ_u	λ_v
CVAE	0.1	10	1	10	0.1	10	0.1	10
CATA	10	0.1	10	0.1	10	0.1	10	0.1

4.5.4 Experimental results

For each dataset, we repeat the data splitting four times with different random splits of training and testing set, which has been previously described in the evaluation methodology section. We use one split as a validation experiment to find the optimal parameters of λ_u and λ_v for our model and CVAE as well. We search a grid of the following values $\{0.01, 0.1, 1, 10, 100\}$ and the best values on the validation experiment have been reported in Table 4.3. The other three splits are used to report the average performance of our model against the baselines. We address the research questions that have been previously defined in the beginning of this section.

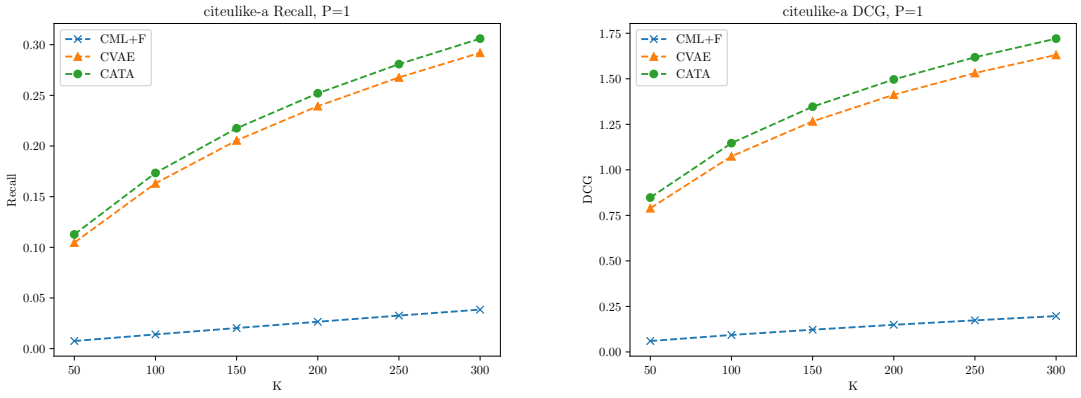
RQ1

To measure the performance of our model against the baselines, we conduct quantitative and qualitative comparisons to answer this question. Figures 4.5, 4.6, 4.7, and 4.8 show the performance of the top-K recommendation under the sparse and dense settings in both datasets. First, the sparse cases are considered a critical challenge for any proposed model since there are less data for training. In the sparse setting where there is only one article in each user's library in the training set, our model, CATA, outperforms the baselines in both datasets in terms of recall and DCG, as figures 4.5 and 4.6 show. More importantly, CATA outperforms the baselines by a

wide margin in Citeulike-t, where it's actually sparser and contains less contextual data. This validates the robustness of our model against the data sparsity.

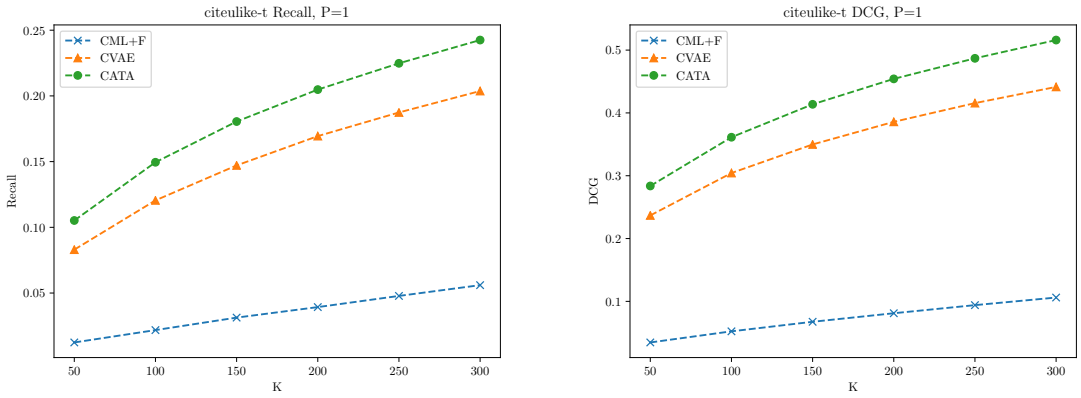
Second, with the dense setting where there are more articles in each user's library in the training set, our model performs comparably to other baselines, as figures 4.7 and 4.8 show. As a matter of fact, many of the existing models actually work well under this setting, but poorly under the sparse setting. For example, CML+F achieves a competitive performance on the dense data; however, it fails on the sparse data since their metric space needs more interactions for users to capture their preferences. As a result, this experiment demonstrates the capability of our model in making more relevant recommendations under both sparse and dense data conditions.

In addition to the previous quantitative results, some qualitative results are reported in table 4.4 as well. The table shows the top 10 recommendations of our model, CATA, against the state-of-the-art model, CVAE, on one selected random user under the sparse setting using Citeulike-a dataset. In this example, $user50$ has only one article in his training library with a title "*Machine learning*". Our model can identify this user interests, and then recommends more general articles based on the same topic "*Machine learning*" rather than recommending more specialized and deeper articles, like the ones recommended by CVAE. For example, the recall@10 of CATA in this example equals $\frac{3}{22} = 0.136$, by assuming the number of relevant articles for this users is 22 articles, while DCG@10 equals $\frac{1}{log_2(4)} + \frac{1}{log_2(7)} + \frac{1}{log_2(11)} = 1.146$. From this example and other users' examples we have examined, we can state that our model detects the major elements of articles' contents and users' preferences more accurately.



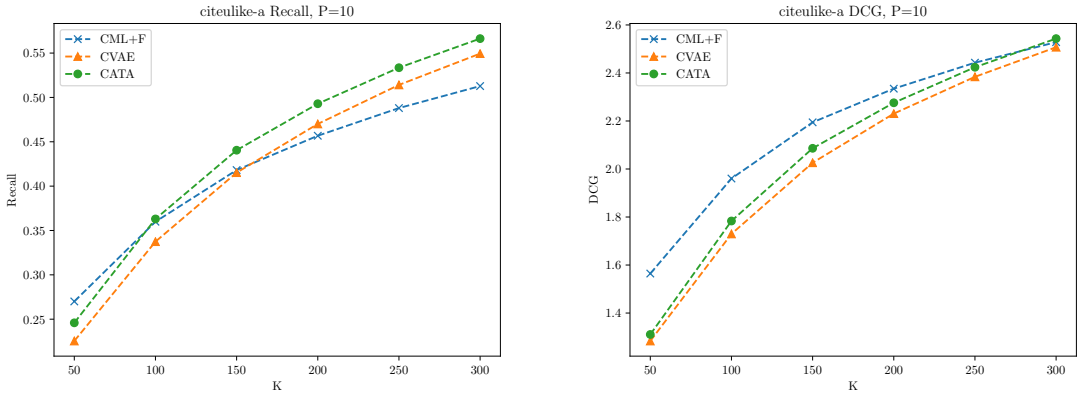
(a) Citeulike-a recall (b) Citeulike-a DCG

Figure 4.5: The top-K recommendation performance under the sparse setting, $P = 1$, for Citeulike-a dataset.



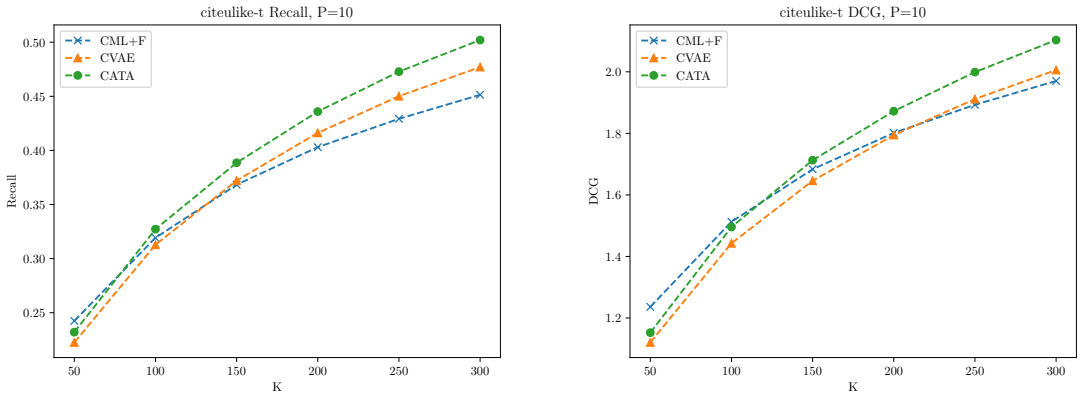
(a) Citeulike-t recall (b) Citeulike-t DCG

Figure 4.6: The top-K recommendation performance under the sparse setting, $P = 1$, for Citeulike-t dataset.



(a) Citeulike-a recall (b) Citeulike-a DCG

Figure 4.7: The top-K recommendation performance under the dense setting, $P = 10$, for Citeulike-a dataset.



(a) Citeulike-t recall (b) Citeulike-t DCG

Figure 4.8: The top-K recommendation performance under the dense setting, $P = 10$, for Citeulike-t dataset.

Table 4.4: An example of the top-10 recommendations of our model (CATA) compared to the CVAE model using the Citeulike-a dataset under the sparse setting.

User ID: 50	
Articles in training set: Machine learning	
CATA	In user library?
1. Data mining practical machine learning tools and techniques	No
2. The visual display of quantitative information	No
3. Pattern recognition and machine learning information science and statistics	**Yes**
4. C programs for machine learning	No
5. Learning opencv computer vision with the opencv library	No
6. Support vector networks	**Yes**
7. Active learning for natural language parsing and information extraction	No
8. C programs for machine learning morgan kaufmann series in machine learning translation approach...	No
9. Intelligence without representation	No
10. The elements of statistical learning	**Yes**
CVAE	In user library?
1. Data mining practical machine learning tools and techniques	No
2. Toward machine emotional intelligence analysis of affective physiological state	No
3. Activity recognition from userannotated acceleration data	No
4. Extracting product features and opinions from reviews	No
5. Active learning for natural language parsing and information extraction	No
6. Learning opencv computer vision with the opencv library	No
7. Named entity extraction from speech	No
8. Face recognition by humans nineteen results all computer vision researchers should know about	No
9. Design experiments theoretical and methodological challenges in creating complex intervention...	No
10. Named entity recognition through classifier combination	No

Table 4.5: Performance comparisons on sparse data with using attention layer (CATA) and without (CATA–). Best values are marked in bold.

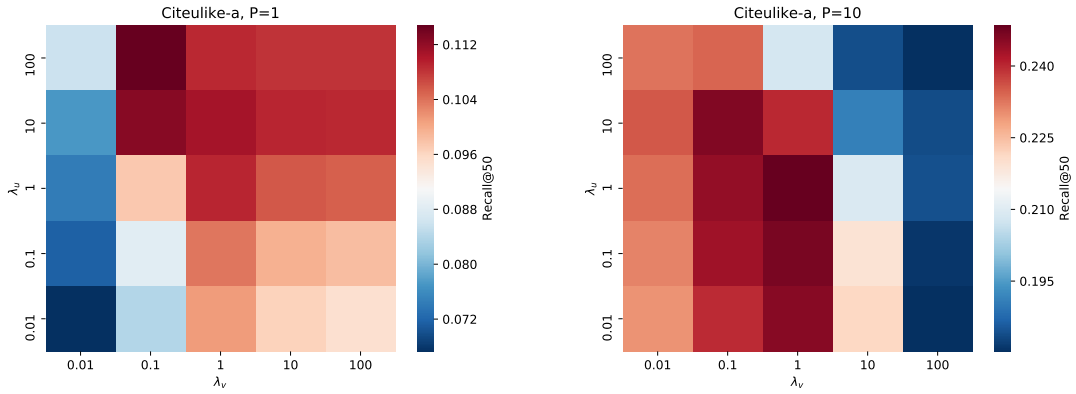
Approach	Citeulike-a		Citeulike-t	
	Recall@300	DCG@300	Recall@300	DCG@300
CATA–	0.3003	1.6644	0.2260	0.4661
CATA	**0.3060**	**1.7206**	**0.2425**	**0.5160**

RQ2

To examine the importance of adding the attention layer into our autoencoder, we create another variant of our model that has the same architecture, but lacks the attention layer, which we call CATA–. We evaluate this model on the sparse datasets, and the performance comparisons are reported in Table 4.5. As the table shows, adding the attention mechanism boosts the performance. Consequently, using the attention mechanism gives more focus to some parts of the encoded vocabularies in each article to better represent the contextual data, eventually leading to increased recommendation quality.

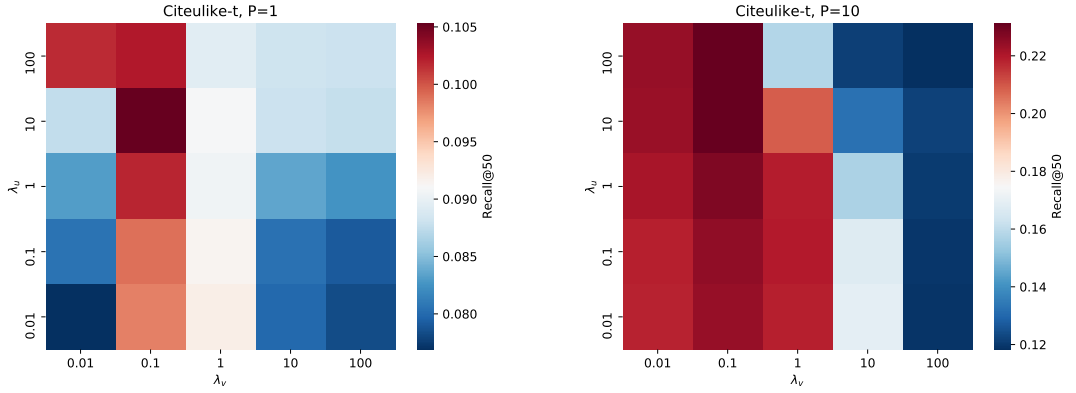
RQ3

As we mentioned before, there are two regularization parameters, λ_u and λ_v, that are used in the objective function of the matrix factorization model to prevent the magnitude of the latent feature vectors from being too large, which eventually prevents the model from overfitting the training data. Our previously reported results were obtained by setting λ_u and λ_v to the numbers in Table 4.3 according to the validation experiment. However, we perform multiple experiments to show the impact of different values of λ_u and λ_v and how they affect the performance of our model. We use different values to set the parameters from the following range {0.01, 0.1, 1, 10, 100}. Figure 4.9 visualizes how our model performs under each combination of



(a) Citeulike-a, P=1

(b) Citeulike-a, P=10



(c) Citeulike-t, P=1

(d) Citeulike-t, P=10

Figure 4.9: The impact of λ_u and λ_v on CATA performance for (a-b) Citeulike-a, and (c-d) Citeulike-t datasets.

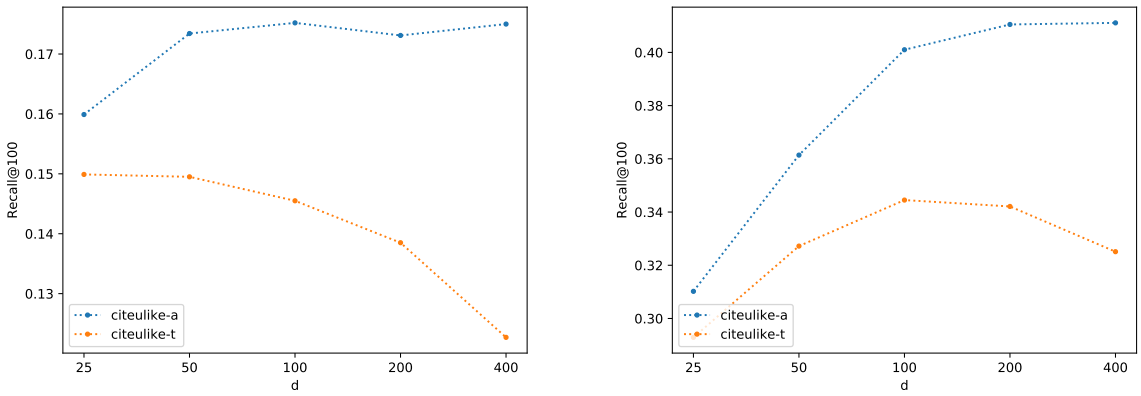
λ_u and λ_v. We find that our model has a lower performance when the value of λ_v is considerably large under the dense setting, as Figures 4.9b and 4.9d show. On the other hand where the data is sparser in Figures 4.9a and 4.9c, a very small value of λ_u (e.g., 0.01) tends to have the lowest performance among all other numbers. Generally, we observe that the optimal performance happens in both datasets when $\lambda_u = 10$ and $\lambda_v = 0.1$. We can conclude that when there is sufficient user feedback, items' contextual information is no longer essential to obtain user preferences, and vice versa.

RQ4

The vectors of the latent features (U and V) represent the characteristic of users and items that a model tries to learn from data. We examine the impact of the size of these vectors on the performance of our model. In other words, we examine how many dimensions in the latent space can represent the user and item features more accurately. It is worth mentioning that our reported results in the RQ1 section use 50 dimensions, which is similar to the size used by the state-of-the-art model (CVAE) in order to have fair comparisons. However, we run our model again using five dimension sizes from the following values {25, 50, 100, 200, 400}. Figure 4.10 shows how our model performs in terms of recall@100 under each dimension size. We observe that increasing the dimension size in dense data leads always to a gradual increase in our model performance, as shown in Figure 4.10b. Also, larger dimension sizes are recommended for sparse data as well. However, they do not necessary improve the model's performance all the time (e.g., the Citeulike-t dataset in Figure 4.10a). Generally, dimension sizes between 100 and 200 are suggested for the latent space dimension.

RQ5

We pretrain our autoencoder first until the loss value of the data converges sufficiently. The loss value shows the error computed by the autoencoder's loss function where it shows how well the model reconstructs outputs from inputs. Figure 4.11 visualizes the number of needed training epochs to render the loss value sufficiently stable. We find that 200 epochs are sufficient for pretraining our autoencoder.



(a) P=1 (b) P=10

Figure 4.10: The performance of CATA model with respect to different dimension values of the latent space under (a) sparse data and (b) dense data.

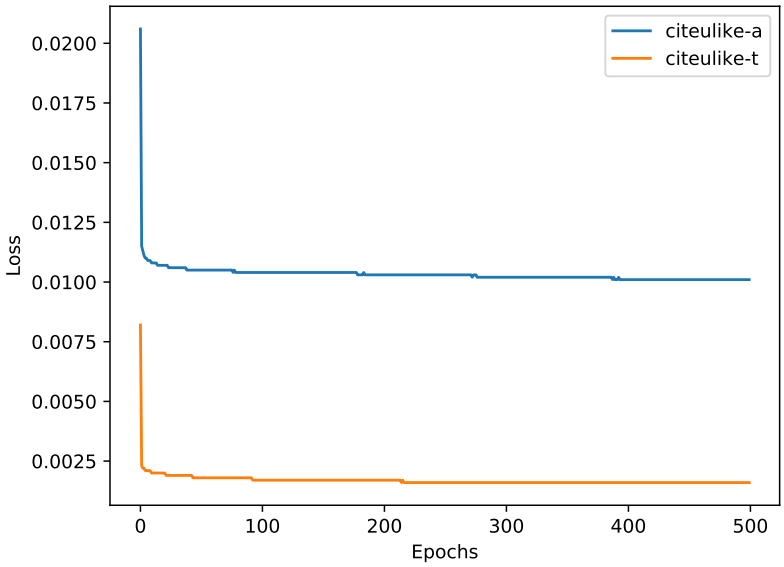


Figure 4.11: The reduction in the loss values vs. the number of training epochs.

4.6 Conclusion

In this work, we propose a Collaborative Attentive Autoencoder (CATA) to leverage side textual information to learn a better compressed representation of the data through an attention mechanism, which can guide the training process to focus on the relevant part of the encoder output in order to improve the model predictions. CATA shows the superiority over the state-of-the-art methods on two scientific article datasets. The performance improvement of CATA increases consistently as the data sparsity increases. The qualitative studies also prove the good quality of our model recommendations.

Chapter 5

CATA++: Leveraging Content Information Independently via Two Separated Attentive Autoencoders

5.1 Abstract

Recommender systems today have become an essential component of any commercial website. Collaborative filtering approaches, and Matrix Factorization (MF) techniques in particular, are widely known for their felicitous performance in recommender systems. However, the natural data sparsity problem limits their performance, where users generally only interact with a small fraction of available items. Consequently, multiple hybrid models have been proposed recently to optimize MF performance by incorporating additional contextual information the MF's its learning process. Although these models improve recommendation quality, there are two primary aspects for further enhancements: (1) multiple models focus only on some portion of the

available contextual information and neglect other portions, and (2) learning the feature space of the contextual information is still a research area that needs further exploration.

In this chapter, we introduce a Collaborative Dual Attentive Autoencoder (CATA++) method that utilizes an item's content and learns its latent space via two parallel autoencoders. We employ the attention mechanism in the middle of our autoencoders to capture the most significant segments of contextual information, which leads to a better representation of the items in the latent space. Also, by leveraging more of the item's content, our model is able to detect more complex patterns that help in disclosing item-item similarities and eventually increasing the quality of recommendations, especially for sparse data. Extensive experiments on three scientific-article datasets have shown that our dual-process learning strategy has significantly improved MF performance in comparison with other state-of-the-art MF-based models using various experimental evaluations. The source code of our methods is available at: https://github.com/jianlin-cheng/CATA.

5.2 Introduction

The amount of data created in the last few years is overwhelming. Interestingly, the data volume grows exponentially yearly compared to the years before, making the era of big data. This motivates and attracts researchers to utilize this massive data to develop more practical and accurate solutions in most of the computer science domains. For instance, recommender systems (RSs) are primarily a good solution to process massive data in order to extract useful information (e.g., users' preferences) to help users with personalized decision making.

There are several applications for recommendation engines such as movies, songs,

books, games, and scientific publications, among other things. For example, scientific paper recommendations are very common applications for RSs. They are quite useful in helping scholars be aware of related work in their research area. Compared to movie and song recommendations, the sparsity of the user-item matrix in the domain of scientific article recommendations is typically higher because the number of available publications greatly exceeds the number of users. For instance, the matrix sparsity in Mendeley[1] is three times higher than it is in Netflix[2] [93].

Existing recommendation models in scientific article domain, such as CDL [34] and CVAE [39], have two major limitations. First, they assume that all features contribute equally to the final prediction. Second, they focus only on some parts of the auxiliary item data and neglect other parts that could also be utilized toward improving the quality of recommendations. In this work, we address the first limitation by using the attention mechanism, while the second limitation is addressed by using two attentive autoencoders in parallel, which are trained separately to detect the item features more accurately. More precisely, we introduce a Collaborative Dual Attentive Autoencoder (CATA++) that is used to recommend scientific papers. We incorporate the attention technique into our deep feature learning procedure to learn from article's textual data (e.g., title, abstract, tags, and citations between papers) in order to enhance the recommendation quality. The features learned by each attentive autoencoder are then employed into the matrix factorization (MF) method for our final articles' suggestions. To show the effectiveness of our proposed model, we perform a comprehensive experiment on three real-world datasets to show how our model works compared to multiple recent MF-based models. The results show that our model can extract better features from the textual data in comparison to

[1] www.mendeley.com
[2] www.netflix.com

other models. More importantly, CATA++ has a higher recommendation quality in the case of high-sparse data. Also, incorporating different parts of the item's data into our training process has a positive effect on our model performance, such that item-item similarities are captured more properly.

The main contributions of this work are summarized in the following points:

- We introduce CATA++, a Collaborative Dual Attentive Autoencoder, that is evaluated on recommending scientific articles. We employ the attention technique into our model, such that only relevant parts of the data can contribute more to item representations. These item representations help in finding similarities among articles.

- We utilize more article content in our deep feature learning process. To the best of our knowledge, our model is the first that utilizes all article content, including title, abstract, tags, and citations, together in one deep model by coupling two attentive autoencoder networks.

- We show the capability of our proposed model of treating different item content independently and then combining their latent features jointly into the matrix factorization method for our ultimate recommendations.

- We evaluate our model using three real-world datasets. We compare the performance of our proposed model with seven baselines. CATA++ achieves superior performance when the data sparsity is considerably high.

The remainder of this chapter is organized in the following order. First, our model, CATA++, is demonstrated in depth in Section 5.3. After that, the experimental results of our model against the state-of-the-art models are discussed thoroughly in Section 5.4. Next, Section 5.5 discusses the theoretical and practical implications of our findings. Lastly, we conclude our work in Section 5.6.

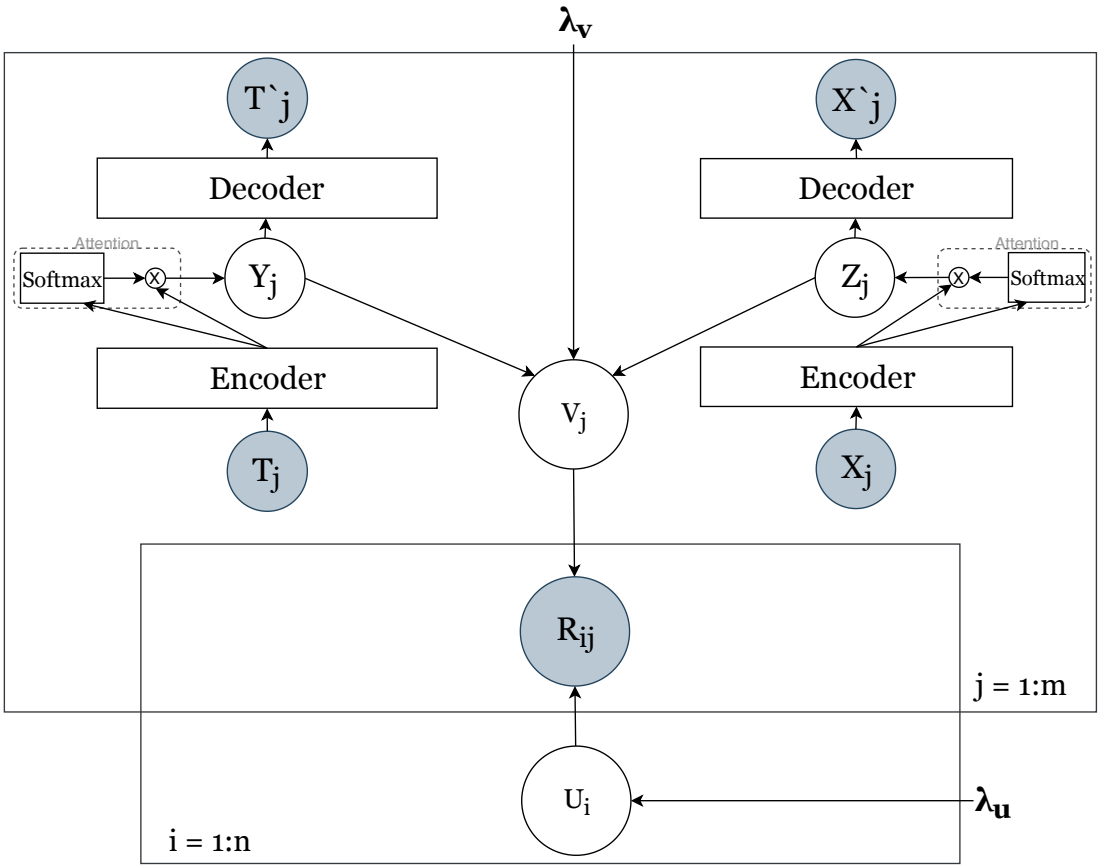


Figure 5.1: Collaborative Dual Attentive Autoencoder (CATA++) architecture.

5.3 Methodology

This chapter introduces an extended model that is built on top of our model in Chapter 4. There is auxiliary contextual information, beside article titles and abstracts, that has never been used by other state-of-the-art methods together in one model, such as article tags (keywords) and citations. Therefore, we extend our previous model, CATA, to include all article content in our model training. We believe that the additional article data (e.g., tags and citations between articles) can be integrated into our model in a way that augments the recommendation quality. We thus name our extended model, CATA++. The architecture of CATA++ is displayed in Figure

5.1.

The first part of our model is our two deep, attentive autoencoders. CATA++ takes two types of input from article content, i.e., $X_j = \{x^1, x^2, ..., x^s\}$ and $T_j = \{t^1, t^2, ..., t^g\}$, where x^i and t^i are real values between zero and one, s is the vocabulary size of our articles' titles and abstracts, and g is the vocabulary size of our articles' tags. In other words, the two types of input from our attentive autoencoder are two normalized bag-of-words histograms that represent the vocabularies of our textual information.

In our previous model (CATA [14]), we train a single attentive autoencoder and incorporate its output into Probabilistic Matrix Factorization (PMF). The objective function of CATA was defined as:

$$\mathcal{L} = \sum_{i,j \in R} \frac{c_{ij}}{2}(p_{ij} - u_i v_j^T)^2 + \frac{\lambda_u}{2} \|u_i\|^2 + \frac{\lambda_v}{2} \|v_j - \theta(X_j)\|^2 \tag{5.1}$$

On the other hand, as Figure 5.1 shows, the latent factors of items (V) from the matrix factorization method are now updated from both autoencoders. CATA++ exploits more of the item's content and trains them via two separate, parallel attentive autoencoders. We use the output of the two separated networks together as the prior information of the items' latent factors from PMF. Therefore, adding a second attentive autoencoder leads to a change of the loss function from our previous chapter (Equation 5.1). The new loss function is defined as follows:

$$\mathcal{L} = \sum_{i,j \in R} \frac{c_{ij}}{2}(p_{ij} - u_i v_j^T)^2 + \frac{\lambda_u}{2} \|u_i\|^2 + \frac{\lambda_v}{2} \|v_j - (\theta(X_j) + \gamma(T_j))\|^2 \tag{5.2}$$

where $\theta(X_j) = Encoder(X_j) = Z_j$, and $\gamma(T_j) = Encoder(T_j) = Y_j$, such that $\theta(X_j)$ and $\gamma(T_j)$ work as the Gaussian prior information for v_j.

To determine the values of u_i and v_j that minimize our previous objective function, we again take the derivative of $\mathcal{L}$ with respect to u_i and v_j. The outputs of the derivatives result in the following equations:

$$\begin{aligned} u_i &= (VC_iV^T + \lambda_u I)^{-1} VC_iP_i \\ v_j &= (UC_jU^T + \lambda_v I)^{-1} UC_jP_j + \lambda_v(\theta(X_j) + \gamma(T_j)) \end{aligned} \tag{5.3}$$

Finally, we use the Alternating Least Squares (ALS) optimization approach to update the values of U and V. ALS works by iteratively optimizing the values of U while the values of V are fixed, and vice versa. This iterative operation continues until the values of U and V converge. The overall steps of our approach are described in Algorithm 2.

Algorithm 2: CATA++ algorithm

```
pre-train first autoencoder with input X;
pre-train second autoencoder with input T;
Z ← θ(X);
Y ← γ(T);
U, V ← Initialize with random values;
while <NOT converge> do
    for <each user i> do
        u_i ← update using Equation 5.3;
    end for
    for <each article j> do
        v_i ← update using Equation 5.3;
    end for
end while
for <each user i> do
    scores_i ← u_i V^T;
    sort(scores_i) in descending order;
end for
Evaluate the top-K recommendations;
```

Table 5.1: Description of CiteULike datasets including tags and citations data.

Dataset	#Users	#Articles	#Pairs	#Tags	#Citations	Sparsity%
Citeulike-a	5,551	16,980	204,986	46,391	44,709	99.78%
Citeulike-t	7,947	25,975	134,860	52,946	32,565	99.93%
Citeulike-2004-2007	3,039	210,137	284,960	75,721	–	99.95%

5.4 Experiments

This section shows a comprehensive experiment in order to address the following research questions:

- **RQ1**: How does our model perform compared to other state-of-the-art models? Prove with quantitative and qualitative analysis.
- **RQ2**: Are both autoencoders (left and right) cooperating with each other to enhance the recommendation performance?
- **RQ3**: What is the impact of different hyper-parameters' tuning (e.g., dimension of the features' latent space, number of layers inside both of the encoder and the decoder, and the two regularization variables λ_u and λ_v) on the performance of our model?
- **RQ4**: What is the computational complexity of our attentive autoencoder, compared to other existing autoencoders used in the state-of-the-art models?

Before answering the aforementioned research questions, we first present our datasets, our evaluation metrics, and the baseline models against which we evaluate our model.

5.4.1 Datasets

We use three real-world, scientific article datasets to evaluate our model against other state-of-the-art models. All the three datasets are collected from the CiteULike

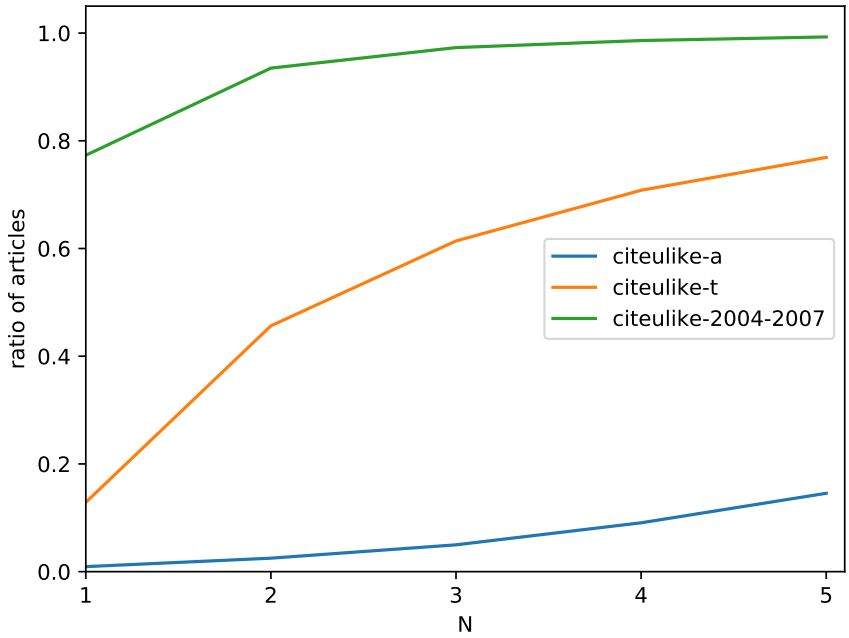


Figure 5.2: Ratio of articles that have been added to $\leq$ N users' libraries.

website. CiteULike was a web service that let users to create their own library of academic publications. The first two datasets are the same datasets we used in our previous chapter. In addition, we add a third dataset in this chapter that is three times larger than the previous ones with regard to the user-article matrix. The data values in this dataset were extracted between 11-04-2004 and 12-31-2007. This dataset is collected by [94] and it has 3039 users, 210137 articles, and 284960 user-article interaction pairs. This dataset is the sparsest among the three datasets, with less than 0.05% of the user-article matrix having interactions. Each user has at least 10 articles in his or her library. On average, each user has 94 articles in his or her library and each article has been added only to one user library. Also, this dataset poses a scalability challenge for recommender systems because of its size.

In this chapter, in addition to the utilization of article titles and abstracts, we

utilize more contextual information such as article tags and citations. Article tags are single-word keywords that have been generated by CiteULike users when they add an article to their library. There are multiple tags assigned to each article. On the other hand, citation data is taken from Google Scholar[3]. At the beginning, we analyze the tags' data and show how many articles assigned to each tag in Figure 5.3. The figure shows the top-20 tags in each dataset. For instance, in Citeulike-t dataset, "bioinformatics" has been assigned to 1522 articles. Using this information, we can catch similarities among articles more accurately. More information about the datasets is shown in Table 5.1.

Figure 5.2 shows the ratio of articles that have been added to five or fewer users' libraries. For instance, 1%, 13%, and 77% of the articles have been added only to one user library in Citeulike-a, Citeulike-t, and Citeulike-2004-2007 datasets, respectively. Moreover, 15% of the articles in Citeulike-a dataset have been added to five or fewer users' libraries. On the contrary, 77% and 99% of the articles in Citeulike-t and Citeulike-2004-2007 datasets have been added to five or fewer users' libraries. From this perspective, Figure 5.2 shows how articles have different degrees of sparsity among the three datasets.

We imitate the same procedure used by state-of-the-art models [34, 4, 39] to preprocess our textual data. For Citeulike-a and Citeulike-t datasets, article titles and abstracts are preprocessed in a similar manner to our previous chapter (Section 4.5.1). For the Citeulike-2004-2007 dataset, we preprocess this dataset such that stop words, non-English words, and English words with less than three letters are removed. Also, words that have a document frequency of more than 0.9 and words that appear in less than three articles are removed as well. Thus, we have 19871 distinct words as our vocabulary list used to build the bag-of-words (BoW) histogram. The created

[3]https://scholar.google.com

Table 5.2: The representation of the article-tag matrix (a) before, and (b) after the matrix is updated when $article_0$ cites $article_1$ and $article_2$ cites $article_0$.

(a) Before integrating the citations data.

	$\mathbf{tag}_0$	$\mathbf{tag}_1$	$\mathbf{tag}_2$	**...**	$\mathbf{tag}_g$
$\mathbf{article}_0$	0	1	0	...	1
$\mathbf{article}_1$	1	0	0	...	0
$\mathbf{article}_2$	0	1	0	...	0
...	...	...	...	...	...
$\mathbf{article}_m$	0	1	0	...	0

(b) After integrating the citations data.

	$\mathbf{tag}_0$	$\mathbf{tag}_1$	$\mathbf{tag}_2$	**...**	$\mathbf{tag}_m$
$\mathbf{article}_0$	1	1	0	...	1
$\mathbf{article}_1$	1	0	0	...	0
$\mathbf{article}_2$	0	1	0	...	1
...	...	...	...	...	...
$\mathbf{article}_m$	0	1	0	...	0

BoW histogram can be described as normalized vectors that has values between zero and one, based on the occurrence counts of the vocabularies. The average number of words per article in both title and abstract after our text preprocessing is 55 words for the Citeulike-2004-2007 dataset.

Simultaneously, we preprocess the tags' information, such that tags assigned to fewer than five articles are removed, and thus we get 7386 and 8311 tags in total for the Citeulike-a and Citeulike-t datasets, respectively. For the Citeulike-2004-2007 dataset, we only keep tags that are assigned to more than 10 articles, and that results in 11754 tags in total for this dataset. After that, we create a matrix for the bag-of-words histogram, $Q \in \mathbb{R}^{m \times g}$, to represent the article-tag relationship, with m being the number of articles and g being the number of tags. This matrix is filled with ones

and zeros, such that:

$$q_{at} = \begin{cases} 1, & \text{if tag}_t \text{ is assigned to article}_a \\ 0, & \text{if otherwise} \end{cases} \tag{5.4}$$

Also, citations between articles are integrated in this matrix, such that if $article_x$ cites $article_y$, then all the ones in vector q_y of the original matrix are copied into vector q_x. We do that to capture article-article relationships. Table 5.2 shows an example of how the article-tag matrix is updated using the citations data. For instance, if $article_0$ cites $article_1$, the article-tag matrix is updated by copying all the ones in $article_1$ into $article_0$. Similarly, if $article_2$ cites $article_0$, all the ones in $article_0$ of the original matrix are copied into $article_2$.

We use recall and normalized Discounted Cumulative Gain (nDCG) as our evaluation metrics to test how our model performs. Unlike the recall metric (which we defined previously in Equation 4.16), the nDCG metric shows the ability of a model to recommend articles at the upper part of the recommendation list. nDCG among all users can be measured using the following equation:

$$nDCG@K = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{DCG@K}{IDCG@K} \tag{5.5}$$

such that:

$$\begin{aligned} DCG@K &= \sum_{i=1}^{K} \frac{rel(i)}{log_2(i+1)} \\ IDCG@K &= \sum_{i=1}^{min(R,K)} \frac{1}{log_2(i+1)} \end{aligned} \tag{5.6}$$

where $|U|$ is the total number of users, i is the rank of the top-K recommended articles, R is the number of relevant articles, and $rel(i)$ is an indicator function that

Table 5.3: Comparison among all models reflecting the data they use in their model training.

Approach	User-article matrix	Side information			
		Title	Abstract	Tags	Citations
POP	✓	–	–	–	–
GMF	✓	–	–	–	–
CML	✓	–	–	–	–
CDL	✓	✓	✓	–	–
CVAE	✓	✓	✓	–	–
CVAE++	✓	✓	✓	✓	✓
CATA	✓	✓	✓	–	–
CATA++	✓	✓	✓	✓	✓

takes the value of one if the article at rank i is a relevant article, and zero if otherwise.

5.4.2 State-of-the-art approaches

We evaluate our approach against the following baselines:

- **POP**: Popular predictor is a non-personalized recommender system. It recommends the most popular articles in a training set, such that all users get identical recommendations. It is widely used as a benchmark for personalized recommendation models.

- **GMF**: Generalized Matrix Factorization (GMF) [46] is a simple non-linear generalization of matrix factorization, where its prediction works as follows: $\hat{y} = sigmoid(W(u_i \odot v_j))$, such that W is the weight and $\odot$ is the element-wise product.

- **CML**: Collaborative Metric Learning (CML) [91] is a metric learning model that pulls items liked by a user closer to him. Recommendations are then generated based on the K-Nearest Neighbor of each user.

- **CDL**: Collaborative Deep Learning (CDL) [34] is a probabilistic model that jointly models both a user-item matrix and an item's content using a stacked denoising autoencoder (SDAE) with PMF.

- **CVAE**: Collaborative Variational Autoencoder (CVAE) [39] is a similar approach to CDL [34]. However, it uses a variational autoencoder (VAE) instead of SDAE to incorporate item content into PMF.

- **CVAE++**: We modify the implementation of CVAE [39] to include two variational autoencoders to engage more item information into the model training, in a manner similar to the way CATA++ functions. By adding another VAE into the model, we change the loss function accordingly, such that the loss of the item's latent variable becomes: $\mathcal{L}(v) = \lambda_v \sum_j \|v_j - (z_j + y_j)\|_2^2$, where z_j is the latent content variable of the first VAE and y_j is the latent content variable of the second VAE.

- **CATA**: Collaborative Attentive Autoencoder (CATA) [14] is our preliminary work that uses a single attentive autoencoder (AAE) to train article content, i.e., title and abstract.

Table 5.3 gives further clarifications about which part of article data is involved in each model training. As the table shows, only CATA++ and CVAE++ use all the available information for their model training.

To have a fair comparison among all baseline models and our model, the authors original code is used to ensure the correct implementation of such models. Also, a validation experiment is used to tune the optimal values of the comparative methods' hyperparameters. For example, Table 5.4 reports the best values of λ_u and λ_v for CDL, CVAE, CVAE++, CATA, and CATA++ according to the validation test. We use a grid search of the values {0.01, 0.1, 1, 10, 100} to obtain their optimal values.

Table 5.4: The parameter settings for λ_u and λ_v for CDL, CVAE, CVAE++, CATA, and CATA++, based on the validation experiment.

Approach	Citeulike-a				Citeulike-t				Citeulike-2004-2007			
	Sparse		Dense		Sparse		Dense		Sparse		Dense	
	λ_u	λ_v	λ_u	λ_v	λ_u	λ_v	λ_u	λ_v	λ_u	λ_v	λ_u	λ_v
CDL	0.01	10	0.01	10	0.01	10	0.01	10	0.01	10	0.01	10
CVAE	0.1	10	1	10	0.1	10	0.1	10	0.1	10	0.1	10
CVAE++	0.1	10	0.1	10	0.1	10	0.1	10	1	10	1	10
CATA	10	0.1	10	0.1	10	0.1	10	0.1	10	0.1	10	0.1
CATA++	10	0.1	10	0.1	10	0.1	10	0.1	10	0.1	10	0.1

In addition, we use the same dimension size across all models; since different dimension sizes are subject to have different results. Also, for all models that are based on matrix factorization, we set a=1, b=0.01, and d=50. In addition, for CDL [34], we set λ_n=1000, λ_w=0.0001, and use a two-layer SDAE network architecture that has a structure of "#Vocabularies-200-50-200-#Vocabularies" to run their code on our datasets. For CVAE [39] and CVAE++, we use a three-layer VAE network architecture, which is similar to the structure reported in their paper [39], with a structure equivalent to "#Vocabularies-200-100-50-100-200-#Vocabularies". Finally, for CATA and CATA++, a four-layer AAE network architecture in the form of "#Vocabularies-400-200-100-50-100-200-400-#Vocabularies" is used train our models.

5.4.3 Experimental results

We now answer the research questions that have been previously defined in the beginning of this section.

RQ1

To show how our model performs, we perform quantitative and qualitative analyses to address this question. Figures 5.4, 5.5, and 5.6 show how our model performs compared to the other models under the sparse setting, $P = 1$, based on the recall and nDCG metrics for all the three datasets. In the same way, Figures 5.7, 5.8, and 5.9 show the same performance under the dense setting, $P = 10$, based on the recall and nDCG metrics as well.

First, the sparse cases are more challenging for any recommendation model, due to the scarce feedback data that is used for the model training. In the sparse setting where there is only one article in each user library for the training phase, CATA++ achieves a superior performance relative to other MF-based models in all datasets in terms of recall and nDCG. More importantly, CATA++ beats the best model among all the baselines, CVAE++, by a wide margin in the Citeulike-2004-2007 dataset, where it is actually the sparsest and contains a huge number of articles. This validates the robustness of our model to work with sparse data. On the other hand, among all baseline models, POP, GMF, and CML generally have the lowest performance, which validates the usefulness of integrating contextual data to alleviate the data sparsity problem.

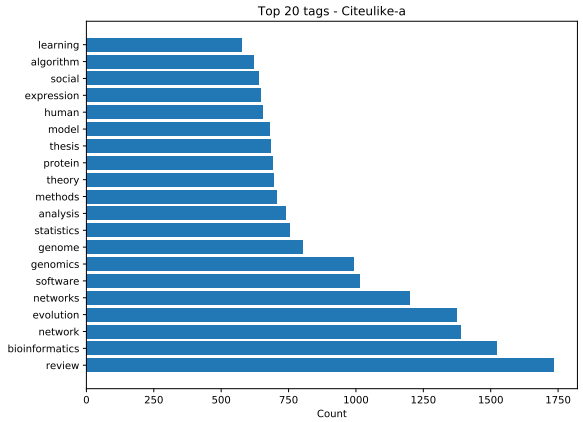
Second, in the dense setting where there are more articles in each user library for the training phase, our model again beats the other models, as Figures 5.7, 5.8, and 5.9 show. In reality, many of the existing models actually work well under this dense setting, but poorly under the sparse setting. For example, CDL fails to beat POP with the Citeulike-t dataset under the sparse setting, and then easily beats POP under the dense setting as Figures 5.5a and 5.8a show. Also, the CML model, which integrates no contextual information, performs very well under the dense setting and has competitive results with other context-aware models, but again performs poorly

on sparse data.

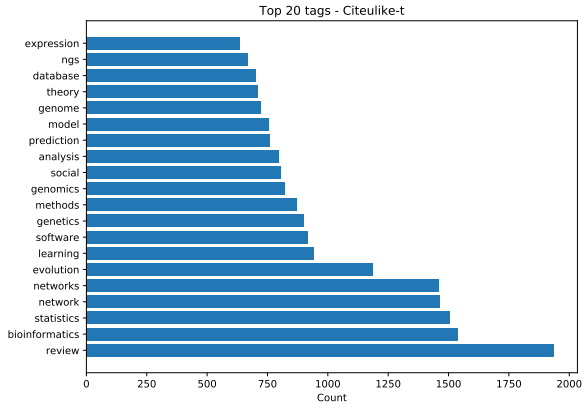
Table 5.7 shows the percentage increase in our model performance, CATA++, over the best competitor among all other baselines. This percentage measures the increase in performance, which can be calculated according to the following formula: $improv\% = (p_{our} - p_{sota})/p_{sota} \times 100$, where p_{our} is the performance of our model, and p_{sota} is the performance of the best model among all baselines. Also, the table shows if the increase in performance is statistically significant, based on the p-value test ($p \leq 0.05$) suggested in [95] for comparing the performance of two models. As the table shows, our model achieves statistically significant improvements over the best model among the baselines in all sparse cases with respect to two different evaluation metrics. For instance, using Recall@5 metric, our model achieves a 14.44%, 28.25%, and 49.21% significant improvement in all sparse cases of Citeulike-a, Citeulike-t, and Citeulike-2004-2007 datasets, respectively. More importantly, the table shows that the improvement in our model performance is remarkably higher with smaller K values. On the other hand, on average, our model achieves a slightly improvement increase in the dense cases, which concludes the robustness of our model to work under sparse and dense data.

In addition to the aforementioned quantitative analysis, qualitative analysis is also reported in Tables 5.5 and 5.6 to show the quality of recommendations using real examples. The first example (i.e., Table 5.5) shows the top-10 recommended articles generated by our model (CATA++) and the other competitive model (CVAE++) for one random user using the Citeulike-2004-2007 dataset under the sparse setting. With this case study, we seek to gain a deeper insight into the difference between the two models through the recommendations they make. The example in the table presents $user2214$, who has only one paper in his or her training set entitled "*A collaborative filtering framework based on fuzzy association rules and multiple-level*

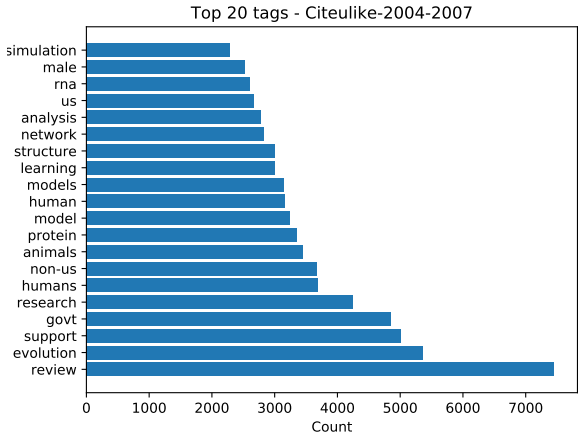
similarity". This example defines the sparsity problem very well where a user has limited feedback data. Based on the article's title, this user is probably interested in recommender systems and more specifically in collaborative filtering (CF). After analyzing the results of each model, we can deduce that our model can recommend more relevant articles than the other baseline. For instance, most of the top-10 recommendations based on CATA++ are related to the user's interest. Our model accuracy in this example is 0.4. Even though CVAE++ generates relevant articles as well, some irrelevant articles could be recommended as well such as the recommended article #7, entitled "*Optimizing search engines using clickthrough data*", which is more about search engines than RSs. In addition, the second example in Table 5.6 presents *user*3830, who has only one paper in his/her training set entitled "*A homology theory for spanning trees of a graph*". Based on the article's title, this user is probably interested in mathematics and more specifically in graphs. The results show again that our model can recommend more relevant articles than the other baseline. For instance, most of the top-10 recommendations based on our model are related to the user's interest, while irrelevant articles could be recommended by CVAE++, such as the recommended article #3, entitled "*Jena: implementing the semantic web recommendations*", which is obviously unrelated to the user's interests. After examining multiple examples, we can conclude that our model identifies user preferences more accurately, especially in the presence of limited data.



(a) Citeulike-a dataset

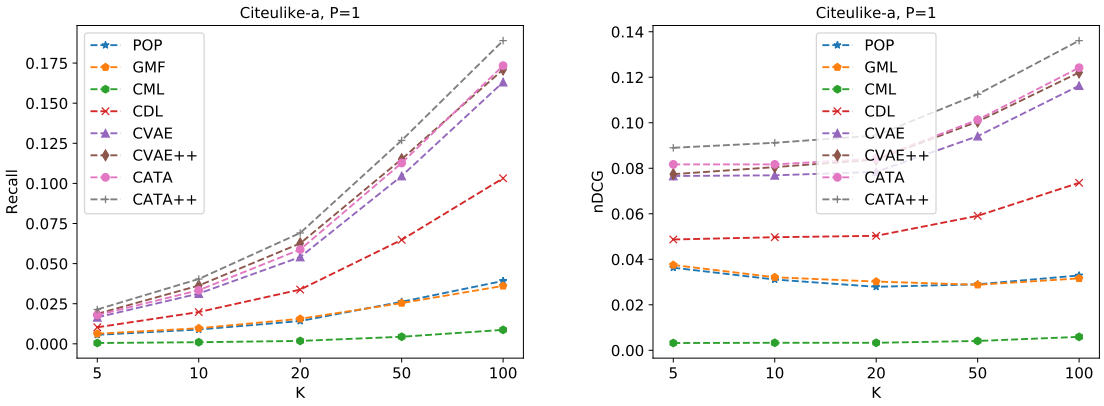


(b) Citeulike-t dataset



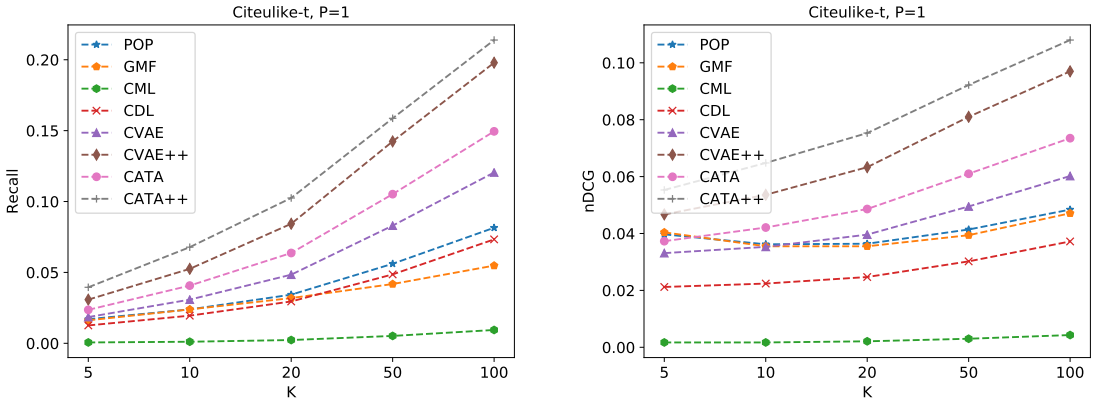
(c) Citeulike-2004-2007 dataset

Figure 5.3: The top-20 tags among all datasets.



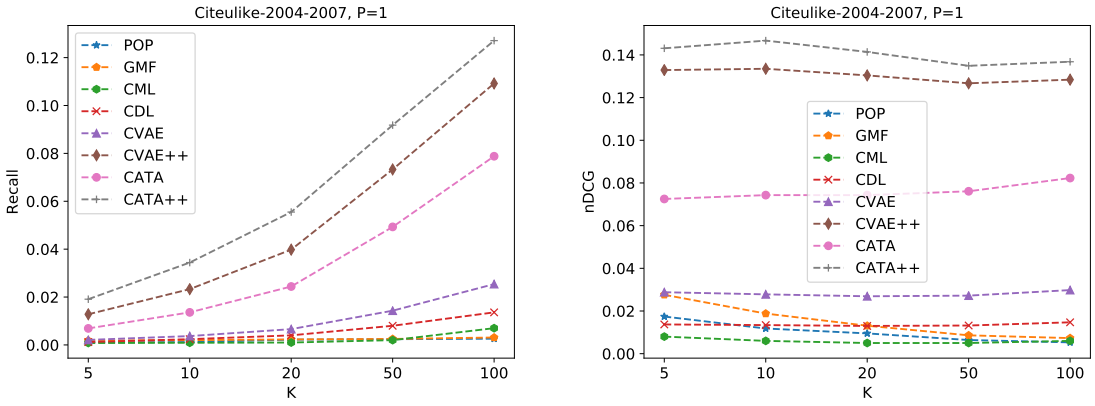
(a) Citeulike-a recall (b) Citeulike-a nDCG

Figure 5.4: The top-K recommendation performance under the sparse setting, $P = 1$, for the Citeulike-a dataset.



(a) Citeulike-t recall (b) Citeulike-t nDCG

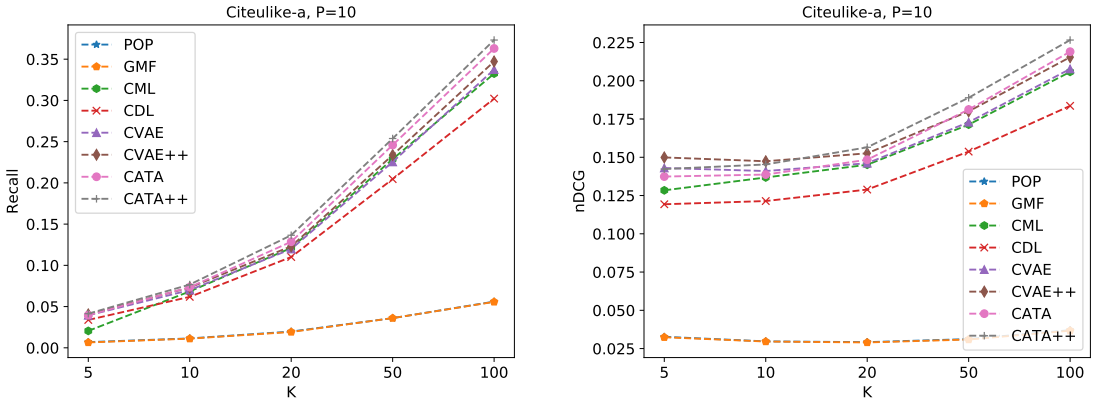
Figure 5.5: The top-K recommendation performance under the sparse setting, $P = 1$, for the Citeulike-t dataset.



(a) Citeulike-2004-2007 recall

(b) Citeulike-2004-2007 nDCG

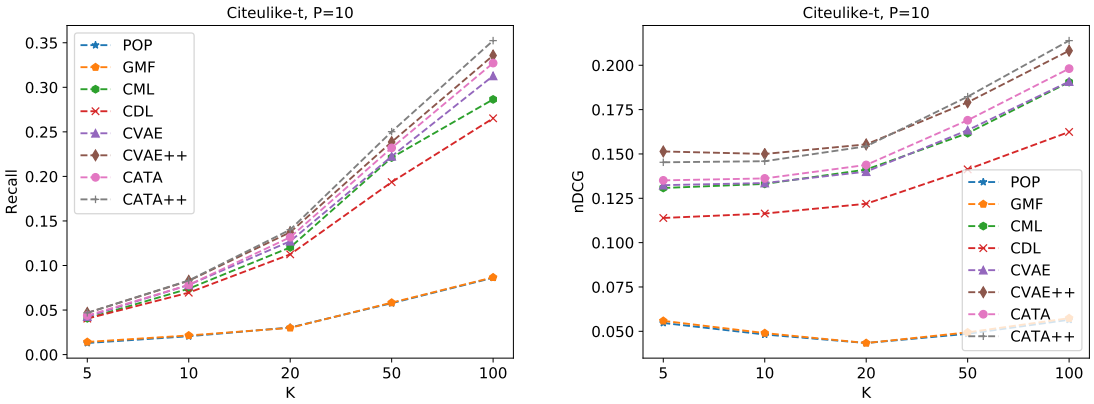
Figure 5.6: The top-K recommendation performance under the sparse setting, $P = 1$, for the Citeulike-2004-2007 dataset.



(a) Citeulike-a recall

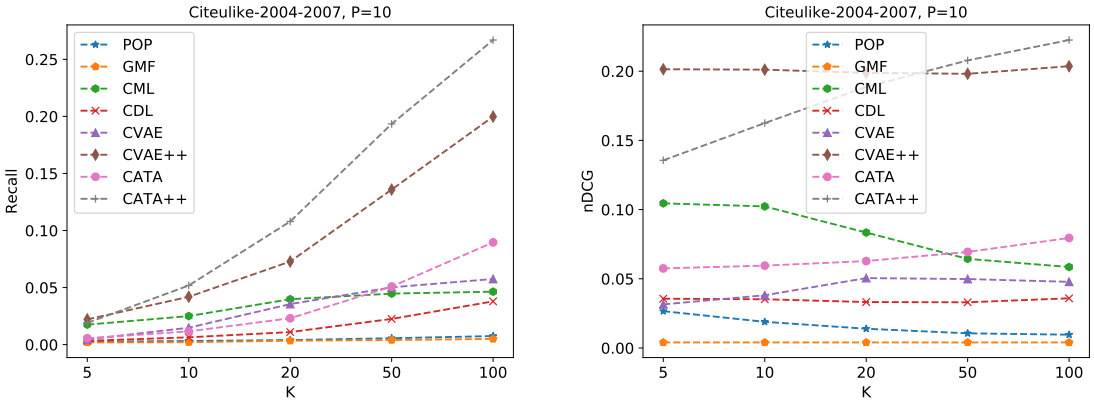
(b) Citeulike-a nDCG

Figure 5.7: The top-K recommendation performance under the dense setting, $P = 10$, for the Citeulike-a dataset.



(a) Citeulike-t recall (b) Citeulike-t nDCG

Figure 5.8: The top-K recommendation performance under the dense setting, $P = 10$, for the Citeulike-t dataset.



(a) Citeulike-2004-2007 recall (b) Citeulike-2004-2007 nDCG

Figure 5.9: The top-K recommendation performance under the dense setting, $P = 10$, for the Citeulike-2004-2007 dataset.

Table 5.5: The first example to show the quality of recommendations using the sparse cases of the Citeulike-2004-2007 dataset.

User ID: 2214	
Articles in training set: A collaborative filtering framework based on fuzzy association rules and multiple-level similarity	
CATA++	In user library?
1. Item-based collaborative filtering recommendation algorithms	No
2. Combining collaborative filtering with personal agents for better recommendations	No
3. An accurate and scalable collaborative recommender	No
4. Google news personalization: Scalable online collaborative filtering	**Yes**
5. Combining collaborative and content-based filtering using conceptual graphs	**Yes**
6. Link prediction approach to collaborative filtering	No
7. Slope one predictors for online rating-based collaborative filtering	No
8. Slope one predictors for online rating-based collaborative filtering	**Yes**
9. A decentralized CF approach based on cooperative agents	No
10. Toward the next generation of recommender systems: A survey of the state-of-the-art..	**Yes**
CVAE++	In user library?
1. Combining collaborative filtering with personal agents for better recommendations	No
2. Explaining collaborative filtering recommendations	No
3. Google news personalization: Scalable online collaborative filtering	**Yes**
4. Learning user interaction models for predicting web search result preferences	No
5. Item-based collaborative filtering recommendation algorithms	No
6. Enhancing digital libraries with TechLens+	No
7. Optimizing search engines using clickthrough data	No
8. Context-sensitive information retrieval using implicit feedback	No
9. A new approach for combining content-based and collaborative filters	No
10. Combining collaborative and content-based filtering using conceptual graphs	**Yes**

Table 5.6: The second example to show the quality of recommendations using the sparse cases of the Citeulike-a dataset.

User ID: 3830	
Articles in training set: A homology theory for spanning trees of a graph	
CATA++	In user library?
1. Collective dynamics of small-world networks	No
2. The geometry of graphs and some of its algorithmic applications	No
3. The fractal geometry of nature	No
4. Posets and planar graphs	**Yes**
5. Symbolic dynamics generated by a combination of graphs	No
6. Very large graphs	No
7. A multilevel algorithm for partitioning graphs	No
8. The notion of dimension in geometry and algebra	No
9. Depth-first search and linear graph algorithms	No
10. A min-max relation on packing feedback vertex sets	**Yes**
CVAE++	In user library?
1. Spectra of random graphs with given expected degrees	No
2. A simple mincut algorithm	No
3. Jena: implementing the semantic web recommendations	No
4. Formalizing refactorings with graph transformations	No
5. Topological fisheye views for visualizing large graphs	No
6. Interface automata	No
7. Tensor decompositions and applications	No
8. Fast approximation of centrality	No
9. Mining coherent dense subgraphs across massive biological networks for functional discovery	No
10. Genetic networks with canalyzing boolean rules are always stable	No

Table 5.7: The improvement percentage in our model's performance over the best competitor according to Recall@5, Recall@100, nDCG@5, and nDCG@100 for (a) the sparse data and (b) the dense data. A (*) indicates statistical significance on $p \leq 0.05$.

Metric	Citeulike-a		Citeulike-t		Citeulike-2004-2007	
	Sparse	Dense	Sparse	Dense	Sparse	Dense
Recall@5	14.44% *	2.22%	28.25% *	0.2%	49.21% *	–
Recall@100	8.99% *	2.81% *	8.08% *	4.97% *	16.39% *	33.58% *
nDCG@5	8.93% *	–	18.42% *	–	7.67% *	–
nDCG@100	9.58% *	3.47% *	11.34% *	2.74% *	6.54% *	9.28% *

RQ2

To examine if the two autoencoders are cooperating with each other in finding more similarities among users and items, we run multiple experiments to show how each autoencoder performs individually compared to how they perform together. In other words, we compare the performance of training all features independently via two separated, parallel autoencoders (i.e., CATA++) against the performance of training all features combined via a single autoencoder, where we combine X and T features together. In the comparison, we also include the performance of training only the right autoencoder (i.e., CATA), which leverages article titles and abstracts (X), and the performance of training the left autoencoder, which leverages article tags and citations if available (T). Figure 5.10 shows the overall results. As the figure shows, the dual-process strategy always garners better results than the other techniques, where they use a single autoencoder, except in one case in Figure 5.10c. We believe that training X and T separately, like what CATA++ does, can extract more meaningful features than if we combine them together in a single autoencoder.

In addition, the performance of the left autoencoder and the right autoencoder are comparable to one another, such that the right autoencoder is better than the left

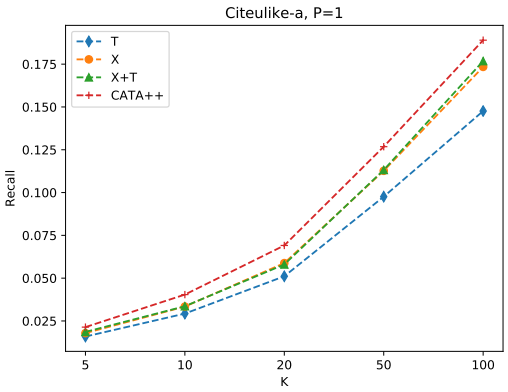
autoencoder using the Citeulike-a dataset, while the left autoencoder is better than the right autoencoder in the other two datasets. We can conclude, by coupling both autoencoders together, our model is able to identify more similarities among users and items, which eventually leads to better recommendations.

RQ3

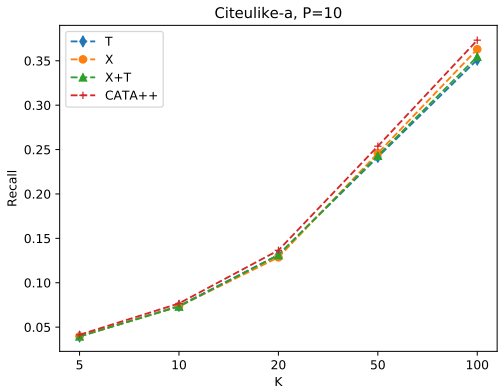
We conduct several experiments to find the influence of tuning some hyper-parameters on the performance of our model, such as the dimension of the latent features, the number of hidden layers of our attentive autoencoder, and the two regularization parameters, λ_u and λ_v, used to learn the user's and article's latent features.

First, the dimension of the latent space used to report our results in the previous section is 50, i.e., each user's and item's latent feature, u_i and v_j, is a vector of size 50. We use the exact number as in the state-of-the-art approach, CVAE, in order to have fair comparisons. However, to see the impact of different dimension sizes, we repeat our whole experiment by changing the size into one of following values: {25, 50, 100, 200, 400}. In other words, we set the size of the latent factors of PMF and the size of the bottleneck of our attentive autoencoder to one of the previous values. As a result, on average, we observe that when the dimension size is equal to 200, our model has the best performance among all three datasets as Figure 5.11a shows. This is because the increase of the latent dimensionality allows the model to detect and learn more of the user and item characteristics. However, there is always a trade-off between the latent dimensionality and the computational complexity. Higher dimensions leads to an increase in training time. Generally, setting the latent space with a size between 100 and 200 is the ideal to gain a remarkable performance, taking into account the computational complexity.

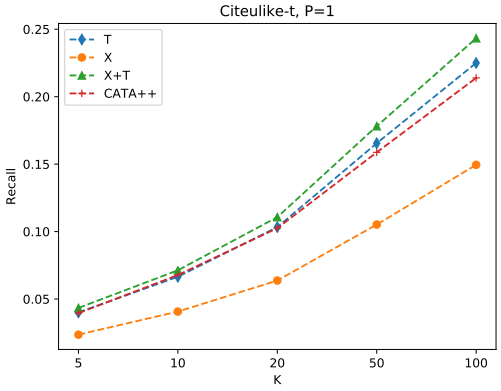
Second, a four-layer network is used to construct our attentive autoencoder when



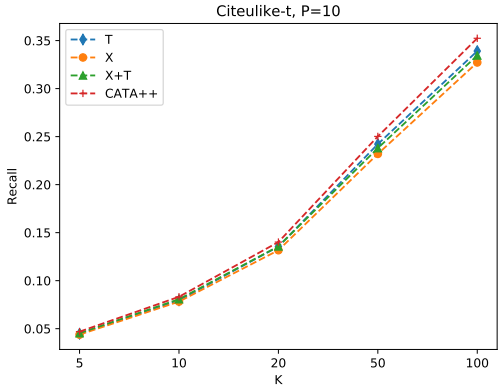
(a) Citeulike-a, P=1



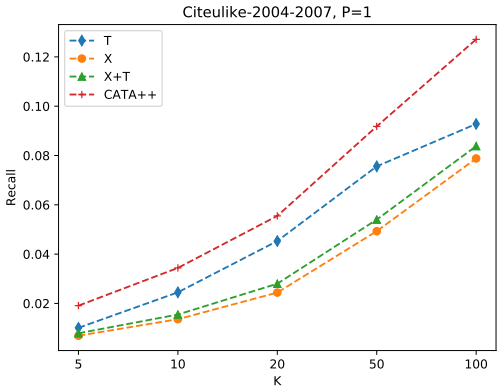
(b) Citeulike-a, P=10



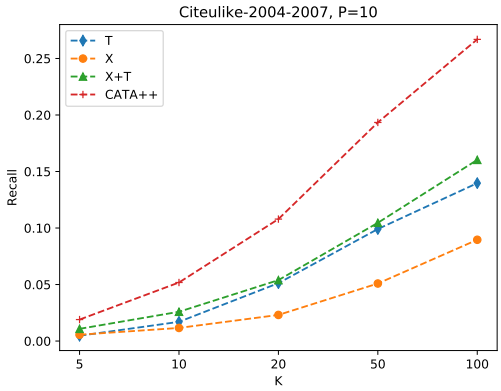
(c) Citeulike-t, P=1



(d) Citeulike-t, P=10

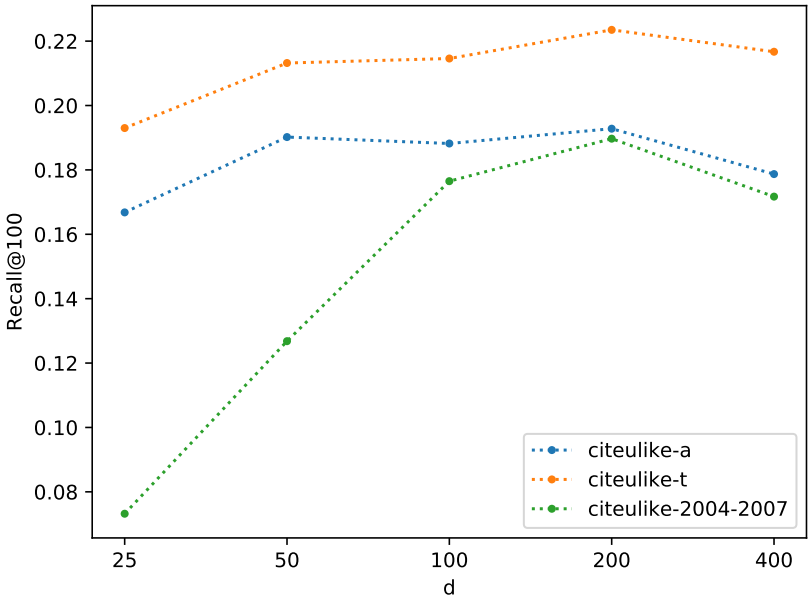


(e) Citeulike-2004-2007, P=1

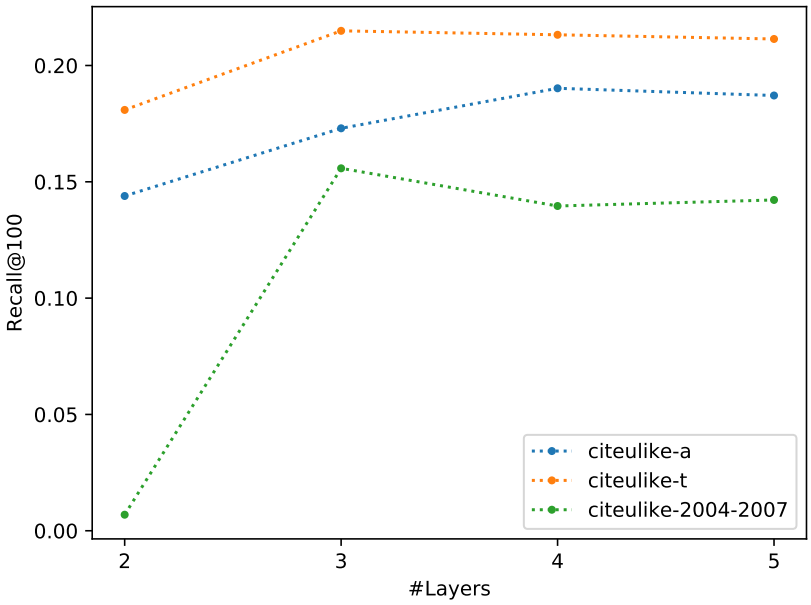


(f) Citeulike-2004-2007, P=10

Figure 5.10: The performance results using only the left autoencoder (T) vs. the right autoencoder (X), compared to combine all features via a single autoencoder (X+T) and train features independently via two separated autoencoders (CATA++) for (a-b) Citeulike-a, (c-d) Citeulike-t, and (e-f) Citeulike-2004-2007 datasets.

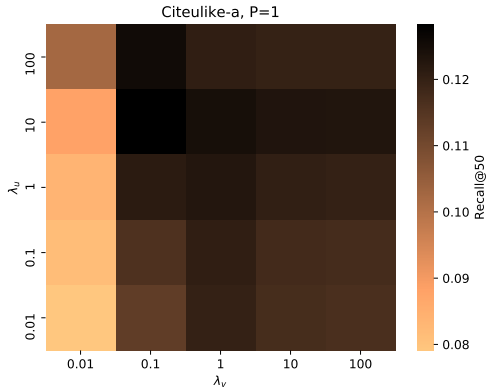


(a) Latent space dimension

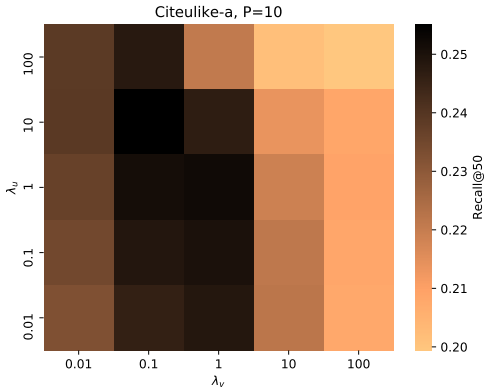


(b) Number of layers

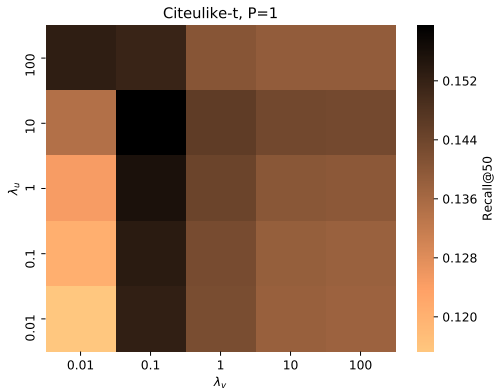
Figure 5.11: The impact of hyper-parameters' tuning on CATA++ performance for: (a) the dimension of features' latent space, and (b) the number of layers inside each encoder and decoder.



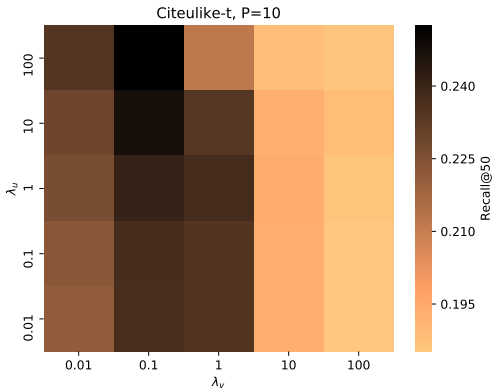
(a) Citeulike-a, P=1



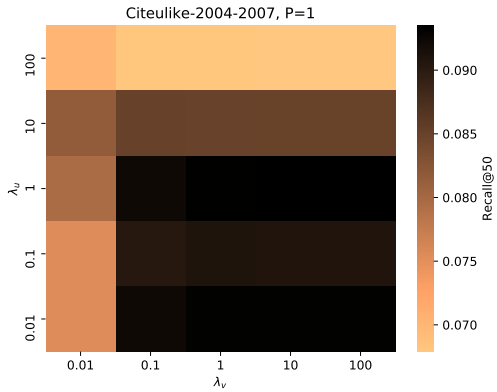
(b) Citeulike-a, P=10



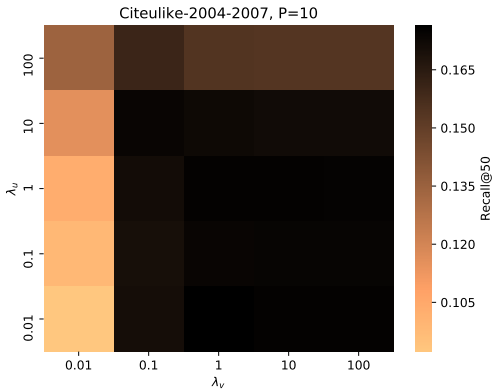
(c) Citeulike-t, P=1



(d) Citeulike-t, P=10



(e) Citeulike-2004-2007, P=1



(f) Citeulike-2004-2007, P=10

Figure 5.12: The impact of λ_u and λ_v on CATA++ performance for (a-b) Citeulike-a, (c-d) Citeulike-t, and (e-f) Citeulike-2004-2007 datasets.

we reported our results previously. The four-layer network has a shape of "#Vocabularies-400-200-100-50-100-200-400-#Vocabularies". However, we again repeat the whole experiment by changing the number of layers starting from two to five layers, such that each layer is half the size of its previous one. As Figure 5.11b shows, using fewer than three layers are not enough to learn from the item's content. This proves the effectiveness of our deep feature learning process to gain more constructive features. Deep neural networks have been proven to be the appropriate mechanism once the training data is considerably big. Generally, three-layer and four-layer networks are sufficiently ideal to train our model.

Third, we repeat our experiment again with setting the values of λ_u and λ_v from the following range: $\{0.01, 0.1, 1, 10, 100\}$. Figures 5.12a and 5.12c show the performance of sparse data using the Citeulike-a and Citeulike-t datasets, respectively. From these two figures, using a lower value of λ_v typically results in a lower performance, meaning the user feedback data is not sufficient and the model needs more article information. The same thing can be said of both scenarios using the Citeulike-2004-2007 dataset in Figures 5.12e and 5.12f. In Figure 5.12e, a higher value of λ_u decreases the performance where user feedback is scarce. Even though Figure 5.12f shows the performance under the dense setting for the Citeulike-2004-2007 dataset, it still represents the sparsity problem with respect to the articles, as we showed previously in Figure 5.2, where 80% of the articles have been only added to one user library. In addition, Figure 5.12c is different than Figures 5.12a, 5.12e, and 5.12f, where increasing λ_v does not necessarily improve the performance. We believe that the missing information from the article content is one reason for that, as we have previously indicated that the average number of vocabularies in this dataset is 19 words compared to 67 words, and 55 words in the Citeulike-a and Citeulike-2004-2007 datasets. On the other hand where user feedback is considerable enough, a

higher value of λ_v results in a lower performance, as Figures 5.12b and 5.12d show.

RQ4

In this section, the scalability of the aforementioned models with respect to running time is examined. In particular, we show the computational complexity of the autoencoder-based models using each one of our datasets. To do that, we use similar settings for all examined models, such that the network structure and the computational resources are similar to each other. Table 5.8 shows the computational complexity (in seconds) for training one epoch among autoencoder-based models using one CPU. As the table shows, CDL, CVAE, and CATA have a comparable running time. More precisely, there is no significant difference in time complexity among the three autoencoders, which are our attentive autoencoder (AAE), the variational autoencoder (VAE) used in CVAE [39], and the stacked denoising autoencoder (SDAE) used in CDL [34].

On the other hand, using more item content for training is expected to make a longer running time. However, For CATA++ and CVAE++, this longer running time in comparison with those model who use a single autoencoder is not significant. More specifically, CATA++ and CVAE++ take approximately less than two times the running time of CATA and CVAE, which results ultimately in a linear growth in time complexity. As a result, using additional autoencoder with additional data content has been validated to scale well with bigger datasets, and more importantly, it remarkably improves the recommendations quality.

Table 5.8: The computational complexity (in seconds) for training one epoch among autoencoder-based models.

Approach	Citeulike-a	Citeulike-t	Citeulike-2004-2007
CDL	40	135	1253
CVAE	50	165	1540
CATA	49	179	1705
CVAE++	100	247	2505
CATA++	98	239	2673

5.5 Discussion

CATA++ is built on top of our previous model, CATA [14]. However, it extraordinary boosts the quality of recommendations by incorporating additional relational article's information into our deep feature learning. Our model shows the capability of learning item features via two separated attentive autoencoders. In particular, we show the beneficial impact of treating different item content independently and then combine their latent features jointly into the matrix factorization approach.

Our experimental evaluations validate the ability of our model to overcome the limitations mentioned in the beginning of this chapter. The current related work hold two shortcomings, i.e., they are not effectively exploiting all available item content in a single model, and also they are treating all features equally, where in reality different features could contribute differently in predictions. For instance, we show that our dual-learning mechanism, used on both CATA++ and CVAE++, have a higher recommendation quality than their original versions (i.e., CATA and CVAE), which emphasizes the usefulness of involving more of the item data in a single model. In addition, our attentive autoencoder (AAE) can extract more constructive information over the variational autoencoder (VAE) and the stacked denoising autoencoder (SDAE), as CATA has the superiority over CVAE and CDL, and CATA++ has the

superiority over CVAE++. This demonstrates the great impact of integrating the attention mechanism into our deep feature learning.

Another influential implication of this work is that our model shows the superiority over multiple competitive models using extremely sparse, real-world datasets. In reality, data sparsity is one of the real problems facing recommender systems. Having a huge amount of available items from different categories, users tend only to interact with a tiny fraction of items, leads to the difficulty of obtaining user preferences. Saying that, we show that CATA++ achieves a statistically significant improvement with extremely sparse data in terms of several evaluation metrics, such as recall, precision, F1 score, and nDCG.

One limitation of this work is that the dot product used in the matrix factorization (MF) violates the triangle inequality property, where it may cause users and item are placed incorrectly in the vector space. For any three items, the triangle inequality is fulfilled once the sum of distance between any two item pairs in the feature space is greater or equal to the distance of the third item pair, such that $d(x, y) \leq d(x, z) + d(z, y)$. Therefore, a potential extension of this work attempts to replace the MF approach with a metric learning-based approach, such as the ones introduced in [91, 96, 97]. More precisely, a potential future work aims to substitute the dot product mechanism with a distance metric function, e.g., Euclidean distance. By doing so, user-user and item-item relationships might be captured more accurately. Another future direction of this work is to investigate the capability of applying this model in another recommendation domain (e.g., movie recommendations), where movie data (e.g., movie reviews and synopsis) can be used for our deep feature learning.

5.6 Conclusion

In this chapter, we alleviate the natural data sparsity problem in recommender systems by introducing a dual-process strategy to learn from an item's textual information by coupling two parallel attentive autoencoders together. The learned item's features are then utilized in the learning process of the matrix factorization (MF). We evaluate our model through an academic article recommendation task using three real-world datasets. The huge gap in the experimental results validates the usefulness of exploiting more item information and the benefit of integrating the attention technique in finding more relevant recommendations, thus boosting the recommendation accuracy. As a result, our model, CATA++, is superior over multiple state-of-the-art MF-based models according to several evaluation measurements. Furthermore, the performance improvement of CATA++ increases consistently as data sparsity increases from one dataset to another. Even though our model has been applied to a ranking predication task with implicit feedback data, it could be used for a rating prediction task with explicit feedback data as well by altering the final loss function.

Chapter 6

Summary and Future Work

6.1 Summary

Although the concept of neural networks has been defined for more than 50 years, the era of deep learning only started in the last decade when a research paper entitled "A fast learning algorithm for deep belief nets" [98] showed an efficient way to train deep neural networks, something not applicable before due to the limitations of computational resources. Another research paper entitled "Learning deep architectures for AI" [99] showed the capability of deep networks to obtain magnificent results on different tasks. More recently, deep learning gets high recognition lately for being the state-of-the-art techniques in various computer science domains like computer vision, natural language processing, speech recognition, and recommender systems.

Therefore, in this dissertation, we have investigated various deep learning techniques to address the data sparsity problem in recommender systems. The data sparsity problem arises when users rate very small number of items, causing a problem for recommender systems to identify the preferences of their users. An effective

solution to this problem is to take the advantage of available user information (e.g., age, country, and occupation) and item content (e.g., movie reviews, paper abstracts, and song lyrics) along with user feedback data to enrich the learning process, in order to identify the key features that eventually will help make user recommendations.

First, we proposed a deep-hybrid collaborative-filtering model, DeepHCF, that combines two deep learning networks together, i.e., Multilayer Perceptron and Convolutional Neural Network. The two deep networks are trained in parallel under the same objective function, using two different inputs to estimate user ratings. The MLP model takes a user's rating history as input and tries to learn the user-item latent vectors, while CNN takes an item's text reviews, which are written by multiple users, as input and tries to detect popular opinions that most likely fit the user. The output of the two sub-models are used by factorization machines to predict the final ratings. This approach is an example of a rating prediction problem.

Second, we proposed a collaborative attentive autoencoder, CATA, that improves the performance of the Matrix Factorization (MF) approach by including the contextual data of items in the learning process. This approach is evaluated on recommending scientific papers. CATA works by initially pretraining paper's textual data (e.g., the title and abstract) via an attentive autoencoder. The learned latent representation is then used in the learning process of MF, such that the difference between the latent factors of items from MF and the learned latent representation from the attentive autoencoder is minimized.

Third, we proposed a collaborative dual attentive autoencoder, namely CATA++, which is built on top of our previous model. CATA++ adds another attentive autoencoder to train more article's data, such that article's similarities are detected more precisely, which promotes the recommendations quality. The two autoencoders are trained independently and are used together to learn items' latent factors from the

matrix factorization. All our proposed models have been evaluated on at least two datasets, and they have shown to have a better recommendation performance over several state-of-the-art models according to different evaluation procedures.

6.2 Future work

For future work, we suggest exploring into the followings points:

1. New metric learning algorithms could be explored to substitute the Matrix Factorization (MF) technique because the dot product in the MF does not guarantee the triangle inequality. For any three items, the triangle inequality is fulfilled once the sum of distance between any two item pairs in the feature space is greater or equal to the distance of the third item pair, such that $d(x, y) \leq d(x, z) + d(z, y)$. By doing so, user-user and item-item relationships might be captured more accurately. In general, any new metric that defines the distance between any two objects should satisfy the following conditions:

$$\begin{aligned} &1.\ d(x,y) \geq 0 \quad \text{(non-negativity)} \\ &2.\ d(x,y) = 0 \iff x = y \quad \text{(identity of indiscernibles)} \\ &3.\ d(x,y) = d(y,x) \quad \text{(symmetry)} \\ &4.\ d(x,y) \leq d(x,z) + d(z,y) \quad \text{(triangle inequality)} \end{aligned} \tag{6.1}$$

   There are several recent works that present new metric learning methods, which experimentally outperform the MF-based methods. Example of such methods are Collaborative Metric Learning (CML) [91], Latent Relational Metric Learning (LRML) [100], Collaborative Translational Metric Learning (TransCF) [96], Factorized Metric Learning (FML) [101], Collaborative Preference Embedding

(CPE) [102], Symmetric Metric Learning (SML) [97], and Hyperbolic Metric Learning (HyperML) [103]. Table 6.1 shows a comparison among the aforementioned metric learning approaches with respect to two major aspects, i.e., the scoring function (a.k.a. the distance function) and the loss function. The scoring function is a function that measures the distance between two objects.

2. Other language models could be investigated to replace the attentive autoencoder and the convolutional neural network to train the auxiliary information. For example, capsules networks (CapsNet) [104, 105, 106] have reached the performance of the state-of-the-art models. Some have even surpassed the state-of-the-art performance in some areas, especially that of computer vision. CapsNet is a new type of neural network that takes control over CNN. CNN has some drawbacks; for example, the precise location of objects in an image are lost, and images should be trained with different angles in order to be recognized. These well-known issues have been solved by capsules networks by using a vector to represent the properties of objects instead of using a single point like CNN. CNN at its beginning was designed for computer vision problems. Since this point, however, it has been successfully applied in natural language processing problem. Similarly, we believe that CapsNet can be used in a way to train items' contextual information, similarly like what CNN does.

3. Although our focus in this dissertation has been to evaluate our proposed models against the data sparsity problem, other recommendation challenges could be evaluated as well, including the cold start problem and the scalability problem. The cold start problem is very related to the sparsity problem. However, it is only concerned with recommendations for new users who have no feedback data yet in the training data. On the other hand, the scalability problem challenges

researchers to design efficient models that can handle very large datasets. The scalability problem can be evaluated against the computation time needed for training and testing.

4. User data can be gathered and then used to update users' latent factors in the same way that we update items' latent factors in CATA and CATA++. Even though user data is often not available due to the privacy concerns (e.g., CiteU-Like datasets do not have user data), we believe that item data, together with user-item interaction data, can be used to infer user information. For example, we could infer users' interests from tags information assigned to documents. Similarly, as we did in creating the article-tag matrix, we can build a user-tag matrix and then learn users' preferences to update users' latent factors.

5. As we have seen in the previous chapters, auxiliary contextual information is very important to increase recommendations accuracy. However, when splitting user ratings into training and testing splits, we do not take the timestamp into consideration. Due to the fact that users' interests might change over time, evaluating recommender systems in this way might not reflect real-world scenarios. Therefore, timestamps are important features that have been integrated in time-aware recommendation systems. We have not involved time parameters into any of our model training. Thus, we could adjust our model architecture accordingly and evaluate our models as time-aware models.

Table 6.1: A comparison among different metric learning approaches regarding the scoring function and the loss function.

Approach	**Scoring function**	**Loss function**
CML [91]	$d(u,v) = \|u - v\|_2^2$	$\sum_{u,v \in S} \sum_{u,\hat{v} \notin S} \left[d(u,v) - d(u,\hat{v}) + m \right]_+$
LRML [100]	$d(u,v) = \|u + r_{uv} - v\|_2^2$	$\sum_{u,v \in S} \sum_{u,\hat{v} \notin S} \left[d(u,v) - d(u,\hat{v}) + m \right]_+$
TransCF [96]	$d(u,v) = \|u + r_{u^{nbr} v^{nbr}} - v\|_2^2$	$\sum_{u,v \in S} \sum_{u,\hat{v} \notin S} \left[d(u,v) - d(u,\hat{v}) + m \right]_+$
FML [101]	$d(u,v) = \|u - v\|_2^2$	$\sum_{u,v \in R} C_{uv} (Y_{uv} - d(u,v))^2$
CPE [102]	$d(u,v) = \|u - v\|_2^2$	$\sum_{u,v \in S} \sum_{u,\hat{v} \notin S} \left[\big(d(u,\hat{v}) - d(u,v) \big) - m \right]_+ + \left[m - \big(d(u,\hat{v}) - d(u,v) \big) \right]_+$
SML [97]	$d(u,v) = \|u - v\|_2^2$	$\sum_{u,v \in S} \sum_{u,\hat{v} \notin S} \left[d(u,v) - d(u,\hat{v}) + m_u \right]_+ + \left[d(u,v) - d(v,\hat{v}) + n_v \right]_+$
HyperML [103]	$d(u,v) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c} \, \|-u \oplus_c v\|_2^2)$	$\sum_{u,v \in S} \sum_{u,\hat{v} \notin S} \left[d(u,v) - d(u,\hat{v}) + m \right]_+$

Bibliography

[1] S. Sagiroglu and D. Sinanc. Big data: A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 42–47. IEEE, 2013.

[2] F. Yuan. *Learning implicit recommenders from massive unobserved feedback*. PhD thesis, University of Glasgow, 2018.

[3] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54. ACM, 2018.

[4] C. Wang and D. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 448–456. ACM, 2011.

[5] J. Zhang, C. Chow, and Y. Zheng. Orec: An opinion-based point-of-interest recommendation framework. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1641–1650. ACM, 2015.

[6] X. Wang and Y. Wang. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 627–636. ACM, 2014.

[7] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[8] C. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13, 2016.

[9] I. MacKenzie, C. Meyer, and S. Noble. How retailers can keep up with consumers. https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers, 2013. (Accessed on 06-13-2019).

[10] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[11] R. Pan, Y. Zhou, B. Cao, N. N Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *2008 Eighth IEEE International Conference on Data Mining*, pages 502–511. IEEE, 2008.

[12] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, volume 8, pages 263–272. Citeseer, 2008.

[13] M. Alfarhood and J. Cheng. Deephcf: A deep learning based hybrid collaborative filtering approach for recommendation systems. In *2018 17th IEEE In-*

ternational Conference on Machine Learning and Applications (ICMLA), pages 89–96. IEEE, 2018.

[14] M. Alfarhood and J. Cheng. Collaborative attentive autoencoder for scientific article recommendation. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 168–174. IEEE, 2019.

[15] M. Alfarhood and J. Cheng. Deep learning-based recommender systems. In *Deep Learning Applications, Volume 2. Advances in Intelligent Systems and Computing*, pages 1–23. Springer Singapore, 2021.

[16] M. Alfarhood and J. Cheng. Cata++: A collaborative dual attentive autoencoder method for recommending scientific articles. In *IEEE Access*, pages 183633–183648. IEEE, 2020.

[17] J. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[19] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[20] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

[21] L. Jing, P. Wang, and L. Yang. Sparse probabilistic matrix factorization by laplace distribution for collaborative filtering. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[22] C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[23] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[24] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[25] B. Betru, C. Onana, and B. Batchakui. Deep learning methods on recommender system: A survey of state-of-the-art. *International Journal of Computer Applications*, 162(10):17–22, 2017.

[26] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[27] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[28] Z. Batmaz, A. Yurekli, A. Bilge, and C. Kaleli. A review on deep learning for recommender systems: challenges and remedies. *Artificial Intelligence Review*, 52(1):1–37, 2019.

[29] A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.

[30] R. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[31] H. Mak, I. Koprinska, and J. Poon. Intimate: A web-based movie recommender using text categorization. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pages 602–605. IEEE, 2003.

[32] H. Wang, B. Chen, and W. Li. Collaborative topic regression with social regularization for tag recommendation. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[33] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[34] H. Wang, N. Wang, and D. Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

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

[36] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[37] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[38] H. Wang, X. Shi, and D. Yeung. Relational stacked denoising autoencoder for tag recommendation. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[39] X. Li and J. She. Collaborative variational autoencoder for recommender systems. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 305–314. ACM, 2017.

[40] D. Liang, R. Krishnan, M. Hoffman, and T. Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698. International World Wide Web Conferences Steering Committee, 2018.

[41] Y. Wu, C. DuBois, A. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162. ACM, 2016.

[42] S. Sedhain, A. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112. ACM, 2015.

[43] S. Zhang, L. Yao, and X. Xu. Autosvd++: An efficient hybrid collaborative filtering model via contractive auto-encoders. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 957–960. ACM, 2017.

[44] S. Li, J. Kawale, and Y. Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 811–820. ACM, 2015.

[45] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016*. ACM, 2016.

[46] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[47] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, and R. Anil. Wide and deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 2016.

[48] Y. Shan, T. Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 255–262. ACM, 2016.

[49] W. Zhang, T. Du, and J. Wang. Deep learning over multi-field categorical data. In *European conference on information retrieval*, pages 45–57. Springer, 2016.

[50] W. Chen, F. Cai, H. Chen, and M. Rijke. Joint neural collaborative filtering for recommender systems. In *ACM Transactions on Information Systems (TOIS)*, pages 1–30. ACM, 2019.

[51] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[52] J. Liu and C. Wu. Deep learning based recommendation: a survey. In *International Conference on Information Science and Applications*, pages 451–458. Springer, 2017.

[53] D. Kim, C. Park, J. Oh, S. Lee, and H. Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016.

[54] J. Liu, D. Wang, and Y. Ding. Phd: A probabilistic model of hybrid deep collaborative filtering for recommender systems. In *Asian Conference on machine learning*, pages 224–239, 2017.

[55] X. Feng, H. Zhang, Y. Ren, P. Shang, Y. Zhu, Y. Liang, R. Guan, and D. Xu. The deep learning–based recommender system "pubmender" for choosing a biomedical publication venue: Development and validation study. *Journal of medical Internet research*, 21(5):e12957, 2019.

[56] L. Zheng, V. Noroozi, and P. Yu. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017.

[57] Y. Gong and Q. Zhang. Hashtag recommendation using attention-based convolutional neural network. In *IJCAI*, pages 2782–2788, 2016.

[58] T. Bansal, D. Belanger, and A. McCallum. Ask the gru: Multi-task learning for deep text recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 107–114. ACM, 2016.

[59] C. Wu, A. Ahmed, A. Beutel, A. Smola, and H. Jing. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 495–503. ACM, 2017.

[60] H. Jing and A. Smola. Neural survival recommender. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 515–524. ACM, 2017.

[61] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 515–524, 2017.

[62] H. Bharadhwaj, H. Park, and B. Lim. Recgan: recurrent generative adversarial networks for recommendation systems. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys)*, pages 372–376, 2018.

[63] D. Perera and R. Zimmermann. Cngan: Generative adversarial networks for cross-network user preference generation for non-overlapped users. In *The World Wide Web Conference*, pages 3144–3150. ACM, 2019.

[64] C. Sun, H. Liu, M. Liu, Z. Ren, T. Gan, and L. Nie. Lara: Attribute-to-feature adversarial learning for new-item recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM)*, pages 582–590, 2020.

[65] Leichtman Research Group. 69% of u.s. households have an svod service. https://www.leichtmanresearch.com/wp-content/uploads/2018/08/LRG-Press-Release-08-27-18.pdf, 2018. (Accessed on 07-13-2019).

[66] X. He and T. Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '17*. ACM, 2017.

[67] S. Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[68] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[69] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Empirical Methods in Natural Language Processing*, page 1746–1751. EMNLP, 2014.

[70] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[71] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[72] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[73] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International conference on algorithmic applications in management*, pages 337–348. Springer, 2008.

[74] G. Salton and M. McGill. *Introduction to modern information retrieval*. McGraw-Hill, Inc, 1983.

[75] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[76] S. Funk. Netflix update: Try this at home. https://sifter.org/ simon/journal/20061211.html, 2006. (Accessed on 11-13-2019).

[77] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[78] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[79] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[80] M. Luong, H. Pham, and C. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[81] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[82] Yogesh Jhamb, Travis Ebesu, and Yi Fang. Attentive contextual denoising autoencoder for recommendation. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 27–34. ACM, 2018.

[83] C. Ma, P. Kang, B. Wu, Q. Wang, and X. Liu. Gated attentive-autoencoder for content-aware recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 519–527. ACM, 2019.

[84] Y. Tay, A. Luu, and S. Hui. Multi-pointer co-attention networks for recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2309–2318. ACM, 2018.

[85] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T. Chua. Attentional factorization machines: learning the weight of feature interactions via attention networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3119–3125. AAAI Press, 2017.

[86] S. Seo, J. Huang, H. Yang, and Y. Liu. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 297–305. ACM, 2017.

[87] X. He, Z. He, J. Song, Z. Liu, Y. Jiang, and T. Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018.

[88] G. E Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[89] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456. ACM, 2015.

[90] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[91] C. Hsieh, L. Yang, Y. Cui, T. Lin, S. Belongie, and D. Estrin. Collaborative metric learning. In *Proceedings of the 26th international conference on world wide web*, pages 193–201. International World Wide Web Conferences Steering Committee, 2017.

[92] R. He and J. McAuley. Vbpr: visual bayesian personalized ranking from implicit feedback. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[93] A. Vellino. Usage-based vs. citation-based methods for recommending scholarly research articles. *arXiv preprint arXiv:1303.7149*, 2013.

[94] A. Alzogbi. Time-aware collaborative topic regression: Towards higher relevance in textual item recommendation. In *BIRNDL@ SIGIR*, pages 10–23, 2018.

[95] P. Tan, M. Steinbach, and V. Kumar. Introduction to data mining. Pearson Education India, 2016.

[96] C. Park, D. Kim, X. Xie, and H. Yu. Collaborative translational metric learning. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 367–376. IEEE, 2018.

[97] M. Li, S. Zhang, F. Zhu, W. Qian, L. Zang, J. Han, and S. Hu. Symmetric metric learning with adaptive margin for recommendation. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[98] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[99] Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[100] Y. Tay, L. Anh, and S. Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 2018 World Wide Web Conference*, pages 729–739. International World Wide Web Conferences Steering Committee, 2018.

[101] S. Zhang, L. Yao, B. Wu, X. Xu, X. Zhang, and L. Zhu. Unraveling metric vector spaces with factorization for recommendation. *IEEE Transactions on Industrial Informatics*, 16(2):732–742, 2019.

[102] S. Bao, Q. Xu, K. Ma, Z. Yang, X. Cao, and Q. Huang. Collaborative preference embedding against sparse labels. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2079–2087, 2019.

[103] L. Tran, Y. Tay, S. Zhang, G. Cong, and X. Li. Hyperml: A boosting metric learning approach in hyperbolic space for recommender systems. In *WSDM*, pages 609–617, 2020.

[104] S. Sabour, N. Frosst, and G. Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[105] S. Sabour, N. Frosst, and G. Hinton. Matrix capsules with em routing. In *6th international conference on learning representations, ICLR*, pages 1–15, 2018.

[106] A. Kosiorek, S. Sabour, Y. Teh, and G. Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems*, pages 15486–15496, 2019.

VITA

Meshal Alfarhood was born and raised in Zulfi, Saudi Arabia. He obtained his Bachelor of Science degree in Computer Science from King Saud University, Riyadh, Saudi Arabia in 2011. He then joined the same department he graduated from with a teaching assistant position. This faculty position gave him the opportunity to receive a scholarship to pursue his graduate studies in the United States. He earned his Master of Science degree in Computer Science from University of Arkansas in 2015. He then joined the University of Missouri-Columbia as a PhD student in 2016 where he joined the Bioinformatics and Machine Learning Laboratory (BML) under the supervision of Prof. Jianlin Cheng. His research interests include Machine Learning, Big Data Analytics, and Recommender Systems. He defended his dissertation and earned his doctoral degree in November 2020.