Complexity evaluation of CNNs in tightly coupled hybrid recommender systems

PATRIK INGVERUD
Complexity evaluation of CNNs in tightly coupled hybrid recommender systems

Komplexitetsanalys av faltningsnätverk i tätt kopplade hybridrekommandationssystem

Patrik Ingverud
pating@kth.se

Degree Programme in Information and Communication Technology
Master's Programme, Computer Science

Supervisor: Pawel Herman
Examiner: Örjan Ekeberg
Abstract

In this report we evaluated how the complexity of a Convolutional Neural Network (CNN), in terms of number of filters, size of filters and dropout, affects the performance on the rating prediction accuracy in a tightly coupled hybrid recommender system. We also evaluated the effect on the rating prediction accuracy for pretrained CNNs in comparison to non-pretrained CNNs.

We found that a less complex model, i.e. smaller filters and less number of filters, showed trends of better performance. Less regularization, in terms of dropout, had trends of better performance for the less complex models. Regarding the comparison of the pretrained models and non-pretrained models the experimental results were almost identical for the two denser datasets while pretraining had slightly worse performance on the sparsest dataset.
I denna rapport utvärderade vi komplexiteten på ett neuralt fältningsnätverk (eng. Convolutional Neural Network) i form av antal filter, storleken på filtren och regularisering, i form av avhopp (eng. dropout), för att se hur dessa hyperparametrar påverkade träffsäkerheten för rekommendationer i ett hybridrekommendationssystem. Vi utvärderade även hur förträning av det neurala fältningsnätverket påverkade träffsäkerheten för rekommendationer i jämförelse med ett icke förtränat neuralt fältningsnätverk.

Resultaten visade trender på att en mindre komplex modell, det vill säga mindre och färre filter, gav bättre resultat. Även mindre regularisering, i form av avhopp, gav bättre resultat för mindre komplexa modeller. Gällande jämförelsen med förtränade modeller och icke förtränade modeller visade de experimentella resultaten nästan ingen skillnad för de två kompaktare dataseten medan förträning gav lite sämre resultat på det glesaste datasetet.
# Table of Contents

Chapter 1  Introduction .................................................................................................................. 1  

1.1  Problem definition .................................................................................................................... 2  
1.2  Scope....................................................................................................................................... 3  
1.3  Thesis outline ............................................................................................................................ 4  

Chapter 2  Background ................................................................................................................... 5  

2.1  Content-based filtering ............................................................................................................ 5  
2.1.1  Content analyser ................................................................................................................ 6  
2.1.2  Profile learner and filtering component ............................................................................. 9  
2.1.3  Advantage and disadvantage of content-based filtering ................................................. 9  
2.2  Collaborative filtering ........................................................................................................... 9  
2.2.1  Collaborative models and recommendation techniques ................................................. 11  
2.2.2  Advantage and disadvantage of collaborative filtering .................................................. 12  
2.3  Hybrid recommender systems .............................................................................................. 12  

Chapter 3  Related work ............................................................................................................... 15  

3.1  Bag-of-word based recommender systems ............................................................................ 15  
3.2  Extensions of bag-of-word-based recommender systems .................................................... 15  
3.3  Word-embedding-based recommender systems with extensions ...................................... 17  
3.4  Recommendation systems with CNNs ................................................................................ 19  
3.5  Architecture evaluation of CNNs ....................................................................................... 20  
3.6  Pre-training of CNNs and RNNs ....................................................................................... 21  

Chapter 4  Method ......................................................................................................................... 23  

4.1  Datasets ................................................................................................................................. 23  
4.2  Model ..................................................................................................................................... 25  
4.3  Examination method ............................................................................................................. 27  
4.4  Evaluation ............................................................................................................................. 28
Table of Contents

Chapter 5  Results .................................................................................................................. 30

5.1  AIV results ......................................................................................................................... 31

5.2  MovieLens-10m ..................................................................................................................... 33

5.3  MovieLens-1m ....................................................................................................................... 35

5.4  Result of pretraining. ............................................................................................................ 37

Chapter 6  Discussion .............................................................................................................. 39

6.1  Limitations ............................................................................................................................ 41

6.2  Ethics ....................................................................................................................................... 42

Chapter 7  Summary and conclusion .......................................................................................... 43

7.1  Summary ............................................................................................................................... 43

7.2  Further work .......................................................................................................................... 44

References .................................................................................................................................... 45

A  Appendix .................................................................................................................................. 1

A.1  Probabilistic Matrix Factorization ......................................................................................... 1

A.2  Probabilistic Matrix Factorization and CNN ................................................................. 1

A.3  One layered convolutional neural network for text .......................................................... 3

A.4  Convolutional autoencoder ............................................................................................... 6

A.5  Optimization methodology ................................................................................................. 9
Chapter 1

Introduction

The growth and variety of information available on the web and the increase of e-business has made it essential for companies to give customers a more personalized experience through recommender systems. Recommender systems emerged as a research area from the classical information systems such as databases and search engines in the mid 1990’s and are essentially a tool which gives suggestions of items that are of interest for a particular user. From the customer’s perspective, a recommender system does not only provide suggestions of items, it also increases the user experience through a better service and ease of use, especially when a database of items is extremely large or if the user lacks experience or competence to evaluate the items personally. From the service provider’s perspective, it may be used to increase number of sold items, sell more diverse items, increase user satisfaction or to better understand what users want [1].

One of the companies that have contributed to research into recommendation engines is Netflix, which organized the famous prize competition from 2006 to 2009 [2]. Netflix understood that if they could improve their existing recommender system they would also increase the company’s total revenue. The dataset for the competition consisted of 100 480 507 ratings from 480 189 users on 17 770 movies and the goal for the contestants were to predict 2 817 131 users’ ratings. The winning team, BellKor's Pragmatic Chaos, made an ensemble of different models and finally won the prize money of $1M in 2009 [3, 4, 5, 6].

In 2015 Netflix released a report, claiming that the combined effect of personalization and recommendations save them more than $1B per year by having a low monthly subscriber churn. They also state that 80% of the streamed movies come from recommendations which shows that good recommender systems are essential for successful businesses [7]. Examples where recommender systems are being used today are: YouTube in recommendation of videos where they stated in
2010 that 60% of all the video clicks from the start pages were due to recommendations [8], Google in recommendation of ads, Spotify in recommendation of music, and Netflix in recommendation of movies and TV-shows. This clearly shows a great demand for good recommender systems in the industry.

1.1 Problem definition

The Netflix prize dataset is a typical example of where collaborative filtering can be used. Collaborative filtering is looking for similarities in user data to recommend items, where one of the key challenges is to improve the rating prediction accuracy. The only information needed in collaborative filtering is which user has liked or rated which items. However, the prediction accuracy decreases with sparser user data, i.e. when ratings are few. With growing number of items in datasets and the assumption that users still only interact with a few items, they become inherently sparser. This makes it difficult to perform collaborative filtering with peak performance. One solution to this is to use content-based filtering where recommendations are based on similarities between items, i.e. recommending similar items to what the user has liked in the past. The benefits from this approach is that recommendations for each user are independent due to that recommendations for a user are only based on the history of the same user. However, such systems only recommend similar items to what a user has liked in the past, e.g. if a user has bought a bicycle (buying can be the same as liking) the user will probably not need more bicycles to be recommended. In this case it would be better to recommend a helmet, for instance.

Another solution to handle the sparsity problem are hybrid recommender systems, which are recommender systems that incorporate content-based filtering abilities into collaborative filtering, and vice versa. The reason to combine them is to keep the benefits from the two approaches while minimizing the drawbacks. Research has shown that hybrid models have an improved accuracy compared to a standard collaborative filtering model, especially for sparse datasets [9, 10, 11, 12].

In the past decade deep learning has made great progress in many domains such as speech recognition, computer vision and Natural Language Processing (NLP). While these have been natural application domains for deep learning, researchers and the industry have worked on expanding the use of deep learning into other applications. Today, different variations of Recurrent Neural Networks (RNN) and Convolutional
Neural Networks (CNN) are the state-of-the-art in NLP and is more often adopted to the field of recommender system due to their ability of effectively capture complex relations. This has led to many new state-of-the-art performances in various recommender system tasks [13].

Kim et al. [9] proposed a tightly coupled hybrid model, a collaborative filtering model with content-based abilities unified into one model, where a CNN is used to extract latent vector representations from item descriptions [9]. Kim et al. [9] believed their model was the first hybrid recommender system using a CNN to extract latent features from text, which also reflects the state-of-the-art performance in terms of rating prediction accuracy. More recently they proposed an improvement of the model in which they introduced statistics to justify the distribution of rated items [10]. The model itself was their main contribution, but they also investigated how the number of filters of the CNN would affect the rating prediction accuracy. However, no experiments were made on the effects of filter sizes.

To the best of our knowledge, no evaluation of filter size has been made for CNNs in hybrid recommender systems. Therefore, we will investigate the effect of both filter size and the number of filters in the CNN on the rating prediction accuracy. Since filter size and number of filters affect the model’s complexity we will also investigate how the regularization, in terms of dropout, on the CNN affects the accuracy in predicting ratings.

Furthermore, it is well known that pretraining of models, either with unsupervised learning or with supervised learning on an auxiliary task, can improve the performance when there is little original data available or if the data is sparse [14]. Pretraining gives the model better initial values such that it can generalize better. So, the secondary aim of the thesis work is to validate the effect on rating prediction accuracy when pretraining of the CNN is performed.

The way to investigate these questions is to compare the rating prediction accuracy for the different models of the CNN.

1.2 Scope
This project will evaluate the complexity of the CNN in terms of different filter sizes, number of filters, dropout rate and how they affect the rating prediction accuracy. Due to many possible variations of CNN architectures, we limit ourselves to only one
convolutional layer. The experiments will be based on the model proposed by Kim et al. [10]. Furthermore, the architecture of the best performing models in terms of the filter size, number of filters and dropout rate will be used as the encoder network in a convolutional autoencoder in order to pretrain weights in an unsupervised manner.

Since datasets, pre-processing and cleaning of data, as well as the choice of a collaborative model will be equal for all experiments, they are believed to have minor impact in comparison of performance of different CNN architectures. Therefore, no major evaluation or quality assessment for these parameters will be made.

Pre-trained word vectors will be used for this task and only word level embeddings from the Global Vectors (GloVe) vocabulary will be considered. A word vector is a vector of numbers which works as a description of the word. The word vectors initially consist of random values but are updated during a training process such that similar words get similar vectors, the GloVe vocabulary is one of such pretrained word vector vocabularies. The disadvantage with word embeddings is that they do not properly handle misspelled words, which means that noise could be more prone to exist compared to using character embeddings. Word embeddings are also corpus specific, which means that a words semantic meaning could differ when trained with different corpuses.

No variations of the latent feature vectors length for items or users will be explored since this is considered to be a factor for the collaborative model and thus not in the scope of this project.

1.3 Thesis outline
In chapter 2 we present the background of recommender systems and the three major techniques for doing recommendations. In chapter 3 we present the related work which mainly consist of relevant advancements in content-based filtering, collaborative filtering and Natural Language Processing. In chapter 4 we present our datasets, models, examination method and evaluation. In chapter 5 we present our results. In chapter 6 we present a discussion about our results. In chapter 7 we present a summary, conclusion and further work. In Appendix A we present a more detailed technical description of the model used in our experiments, i.e. the matrix factorization, a convolutional neural network, the convolutional autoencoder and the optimization methodology.
Chapter 2

Background

2.1 Content-based filtering
Content-based filtering emerged from the information retrieval and information filtering research but is widely used today in many e-commerce platforms. Content-based recommender systems are trying to recommend similar items to what a user has liked in the past [15]. Many people are familiar to this kind of recommendation as “You like this/these product(s) – You may also like this/these”. An example is shown in Figure 1 where a person has bought a can of soda and is recommended to buy a can of beer. The reason could be explained by a similar description of the items, e.g. both cans are red and consist of sparkling liquid.

The recommendation process for content-based filtering is usually performed in three steps: content analyser, profile learner, filtering component where the profile learner and filtering component sometimes are the same depending on the applied technique [15].
2.1.1 Content analyser

The job of the content analyser is to create structured representations, e.g. vectors, of items such that they can be compared in an efficient way. The importance lies in the ability to distinguish what items are similar and what items are dissimilar. Creating item representations can be done with any kind of information that describes an item but since content-based filtering has developed from information retrieval systems the most common historically has been to create representations from text. For example, the Fab recommender system [16] is a web page recommender system that uses the 100 most important words to represent an item, in this case a web page. One of the most common measure to find the most important words is term frequency-inverse document frequency (TF-IDF) [17, 18]. It is represented with the bag-of-word principle in the following way:
Consider these four sentences as document description:

1. “I like this movie, it is not bad at all” – Document 1
2. “I do not like this movie at all, it is bad” – Document 2
3. “This movie is bad” – Document 3
4. “I like this movie” – Document 4

The size of the vocabulary is denoted as $W$ and is, in this case 11 words, i.e. the number of words that has appeared at least once in any of the documents. This could also be explained as the number of rows shown in Table 1. The bag-of-word-representation of the documents is a fixed length vector with the same length as the vocabulary. For every appearance of a word in the document the word count increases by one, as exemplified in Table 1.

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>Document 1</th>
<th>Document 2</th>
<th>Document 3</th>
<th>Document 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>this</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>movie</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bad</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>at</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>all</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>do</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. This is a visualization of the bag-of-words model showing occurrences of words in each document. The table exemplifies the bag-of-words for four different documents: Document 1 – “I like this movie, it is not bad at all”, Document 2 – “I do not like this movie at all, it is bad”, Document 3 – “This movie is bad” and Document 4 – “I like this movie”.

To calculate the term frequency of word $k_i$ in document $d_j$ the following formula is used.

$$ (1) \quad TF_{i,j} = \frac{f_{i,j}}{\sum_{i=0}^{W} f_{i,j}}, $$
where $f_{i,j}$ is the number of times the word appeared in document $d_j$ and the denominator is the number of words occurring in document $d_j$. However, words that appear with high term frequency in many documents are not useful in distinguishing if two items are similar. Therefore, inverse document frequency is used in combination with term frequency and is calculated in the following way:

$$ (2) \quad IDF_i = \log \frac{N}{n_i}, $$

where $N$ is the number of documents and $n_i$ is the number of different documents where word $k_i$ appears. The TF-IDF score for word $k_i$ in document $d_j$ is then calculated as:

$$ (3) \quad w_{i,j} = IDF_i \times TF_{i,j}. $$

As a result of using TF-IDF, the document matrix would look like Table 2 where the words “this” and “movie” are removed since their TF-IDF is 0, thus not contribute with any information.

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>Document 1</th>
<th>Document 2</th>
<th>Document 3</th>
<th>Document 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.012</td>
<td>0.011</td>
<td>0</td>
<td>0.031</td>
</tr>
<tr>
<td>like</td>
<td>0.012</td>
<td>0.011</td>
<td>0</td>
<td>0.031</td>
</tr>
<tr>
<td>it</td>
<td>0.030</td>
<td>0.027</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>is</td>
<td>0.012</td>
<td>0.011</td>
<td>0.031</td>
<td>0</td>
</tr>
<tr>
<td>not</td>
<td>0.030</td>
<td>0.027</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bad</td>
<td>0.012</td>
<td>0.011</td>
<td>0.031</td>
<td>0</td>
</tr>
<tr>
<td>at</td>
<td>0.030</td>
<td>0.027</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>all</td>
<td>0.030</td>
<td>0.027</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>do</td>
<td>0</td>
<td>0.054</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. This is a visualisation of the bag-of-word model with TF-IDF score. Common words get a low score while document specific words get a high score. Each score is calculated by first using formula (1) for calculating the term frequency, then using formula (2) for calculating the inverse document frequency and then using formula (3) for getting the final TF-IDF score.
2.1.2 Profile learner and filtering component
For content-based filtering systems the profile learner takes liked or disliked items given by a user as input and tries to generalize this data in order to construct a user profile. User profiles that are modelled as a machine learning/probabilistic model are also used as the filtering component, since the profile itself is learnt to classify relevant or non-relevant items. For user profiles that are modelled as user vectors an additional component will be used to do the filtering of relevant and non-relevant items.

The LIBRA system [19] uses the model based approach which sees the recommendations as a classification problem and uses a simple Bayes text classifier based on the documents that are represented in a bag-of-word-vector to learn the user profile, in other words it predicts the rating given a set of words. The final prediction from the user profiles in the LIBRA system is binary, i.e. like or dislike.

Another method is to let the average vector of liked items be the user profile which is proposed in [20], where for vector averaging the Rocchio algorithm is used to create a user profile consisting of one vector. Recommendations are then based on vector similarity between the user profile vector and item vectors, using cosine similarity as measure which is the same as the inner product of the two vectors divided by the product of the magnitude of the two vectors.

2.1.3 Advantage and disadvantage of content-based filtering
The advantage of this approach is that each recommendation is independent, i.e. it recommends items based on one user's specific profile and similarities between items. Drawbacks are that no surprising recommendations will be made, i.e. the diversity of the recommendations is very small since recommended items are similar to previously liked items or the user profile which has been set by the user [15].

2.2 Collaborative filtering
The Netflix prize dataset is an example of where collaborative filtering can be used. Collaborative filtering is based upon the assumption that users with similar ratings like similar items, i.e. there is an overlap of users' interests [21]. The fundamentals of collaborative filtering are that one does not need to know any explicit information about the items or the users, the only data that is needed is what items users have rated or liked in the past. One can often see on e-commerce sites “You have liked
these products – Other people have also liked these”. An example of this is shown in Figure 2 where the first person has bought a pizza, a coke and candy while the second person has bought a pizza and a coke. Since they both have bought pizza and coke they are considered to have similar taste which is why the second person is recommended to also buy candy.

Figure 2. Example of collaborative filtering where recommendations are based on similarities between user profiles. The person to the left in the figure has bought a pizza and a coke, while the person to the right in the figure has bought a pizza, a coke and candy. Since they both have bought pizza and coke they are considered to have similar taste which is why the person to the left is recommended to also buy candy.

Another way to see collaborative filtering is as a user-item matrix. In Table 3 an example of a user-item matrix is illustrated, where rows reflect different users and columns different items. The blue highlighted elements are rated items, which could be ordinal, binary or something else indicating a like/dislike, while white elements with 0 indicate that it is unrated. The goal is to predict the ratings of the unrated items. As an example, User 2 and User 5 has rated similarly on Item 1 and Item 4, therefore User 2 might also rate Item 2 similarly to User 5.
Table 3. This is a user-item matrix as a collaborative filtering example. White boxes with 0 are unrated items while blue boxes with a number between 1-5 are rated items. Similarities between user profiles can be found in order to predict a rating for an unrated item.

<table>
<thead>
<tr>
<th></th>
<th>item 1</th>
<th>item 2</th>
<th>item 3</th>
<th>item 4</th>
<th>item 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>user 2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>user 3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>user 4</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>user 5</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

2.2.1 Collaborative models and recommendation techniques

Collaborative filtering can be divided into two groups: memory based techniques and model based techniques [21].

Memory based techniques are essentially neighbourhood models. A neighbourhood model needs to keep the data in memory and search the dataset to find similar user profiles. This differs from the model based techniques, which are trained with a machine learning algorithm to create a model. One of the earlier collaborative recommender systems was the GroupLens system for net news [22]. The technique used in this system is memory based, calculating correlations between user vectors, e.g. row vectors shown in Table 3, in order to do recommendations.

Breese et al. [21] made an empirical analysis comparing different variations of both memory based techniques and model based techniques. The comparison was made for different correlation and similarity measures between user vectors in the memory based technique, while the model based technique consisted of a clustering-based model and a decision tree model based on probabilities.

In 2009 during the Netflix prize competition, matrix factorization techniques were shown to be superior to memory based techniques and have become a dominant methodology within collaborative filtering [23]. Matrix factorization is developed
from Latent Semantic Analysis, a technique which was first used in content-based filtering, which is explained in section 3.2. The reason matrix factorization has become so popular is due to that it creates latent vectors both for items and users and has a simple way of adding additional information to the model, e.g. in the Netflix prize competition where temporal information and demographics was used. Other examples related to our project are [9, 10, 11], where matrix factorization has been successfully implemented in a hybrid recommender system, which is why the technique is also chosen for this project. The basic principle of matrix factorization is that each user and item have its own latent vector. The user latent vector is supposed to describe the user’s preferences while the item latent vector is supposed to describe the features of the item. The inner product between the user vectors and item vectors will try to approximate the rating predictions, where the goal is to fill the empty places in the rating matrix, e.g. the 0’s in Table 3, while minimizing the errors to the already given ratings. A more technical explanation can be found in the Appendix A.1.

2.2.2 Advantage and disadvantage of collaborative filtering
Advantages of collaborative filtering is that it recommends a greater diversity of items compared to content-based filtering, since the recommendations are based on the overlap of users’ interest. Disadvantages are that it does not work well when the user data is sparse, i.e. items have few ratings or likes given by users [24].

2.3 Hybrid recommender systems
Hybrid recommender systems were introduced to handle the disadvantages of collaborative filtering and content-based filtering while keeping the benefits from the two. A hybrid model can therefore be seen as a combination of content-based filtering and collaborative filtering [24]. In our case the main goal is to build a collaborative recommender system and use additional information in terms of item descriptions to justify the lack of user data. The use of item data is the most common way since it is typically more difficult to collect and use detailed information about users due to privacy issues and that users are not always willing to provide information. As an example, see Item 5 in Table 3. Item 5 has only got one rating by User 4, it is therefore difficult to predict how other users would rate that item and it would be even more difficult if that item had zero ratings. This is sometimes called the cold start problem. Suppose that by comparing item descriptions we learn that
Item 5 is similar to Item 1. An example is shown in Figure 3 where Item 1 and Item 5 are two different Pirates of the Caribbean movies, and where the useful ratings are highlighted with green boxes. Since we know that the items are very similar we can make a more qualified prediction for User 2 and User 5 on Item 5 visualized as red boxes.

Figure 3. Visual example of a simplified hybrid recommender system. Useful ratings are shown in green boxes while the predicted ratings are shown in red boxes. Since Item 1 and Item 5 are very similar, which is described in the left of the figure, we can make a more qualified rating prediction for Item 5 even though it only has one previous rating.

Furthermore, there are two main approaches on how to construct a hybrid model:

- **Loosely coupled** which means that collaborative and content-based filtering are implemented separately and their predictions are then combined [24]. An example is the web page recommender system, Fab [16], where the highest scores from the a collaborative and a content-based model are used as recommendation.

- **Tightly coupled** which is a unified model incorporating both collaborative and content-based characteristics [24]. Collaborative Topic Regression [12], Collaborative Deep Learning [11] and Convolutional Matrix Factorization [10] are prime examples.
The benefits in training a tightly coupled model is that it allows a two-way interaction, which means that it automatically balances the power of the learnt features and the ratings from the collaborative filtering model [11], i.e. the learnt features affect predictions meanwhile the predictions affect the learnt features. Though, tightly coupled models require a more careful design and optimization [13], they require less training steps and have shown in previous work to perform better than loosely coupled models on rating prediction accuracy [11, 12].
Chapter 3

Related work

This chapter is discussing the relevant advancements in collaborative filtering, content-based filtering and NLP with focus on how to represent items and users. The different models are grouped based on the representation technique and in a chronological order to give the reader a better understanding of the advancements to today’s state-of-the-art recommender systems.

3.1 Bag-of-word based recommender systems

Bag-of-word representations with TF-IDF have been used in many recommender systems with success, e.g. in the Fab recommender system [16]. However, even though TF-IDF can filter out non-important words with low weights and give more important words a higher weight, the shortcoming is in representing texts in a bag-of-words principle. The reason is that it does not take word order into account, does not know the context in which the words appear and cannot distinguish similarities between words since words are represented by a scalar weight value. Firstly, this means that two documents that have similar meaning but use different words are interpreted as not being similar, which is the problem of synonyms. Secondly, two documents with different meaning could end up having very similar or even the same latent item vector representation, which is the problem of polysemy (from the vectors perspective). As an example, document 1 and document 2 in Table 2 have similar vector representations but their actual meanings are very different. A lot of research has been done in NLP and recommender systems to produce enhanced latent representations of words and text, which better handle the problem of synonyms and polysemy. However, TF-IDF can still be used to filter out non-important words in order to reduce the vocabulary size.

3.2 Extensions of bag-of-word-based recommender systems

To handle the problem of synonyms Degemmis et al. [25] proposed a content-based filtering system using a synonym dictionary called WordNet [26], where words are
mapped to a group of similar words. Instead of using the traditional bag-of-words model, bag-of-synsets is proposed [25]. To make recommendations one naïve Bayes text classifier model per user, with binary output (recommend/not recommend), is trained on the items a user has liked and disliked in the past.

Another method of finding similarities in words and documents is with Latent Semantic Analysis (LSA) [27] or Probabilistic Latent Semantic Analysis (pLSA) [28]. It is based upon singular value decomposition (matrix decomposition) such that each word and document have their own latent vectors. The latent vectors are learnt by approximating the inner product of the latent word and document vectors to be equal to the TF-IDF score (or equivalent value) for the corresponding word in the corresponding document in the word-document matrix. The latent document vectors can then be used to find similarities by comparing them with the cosine similarity measure. LSA and pLSA were first introduced in query search and content-based filtering to find relevant documents [27, 28] but were later refined and used in collaborative filtering, where probabilistic matrix factorization (PMF) is most popular today [23, 29, 30].

Another method closely related to pLSA is Latent Dirichlet Allocation (LDA) [31], which tries to model each document into one or more topics. The number of topics is an arbitrarily predefined number and the method is based on the assumption that documents are covered by a small set of topics and that topics are covered by a small set of specific words. LDA learns the topic distribution of documents, which can be seen as latent item vectors, and is used as the content-based filtering part in the tightly coupled hybrid recommender model called Collaborative Topic Regression, proposed by Wang et al. [12]. The collaborative part of the model uses PMF to predict ratings.

Later, Wang et al. [11] proposed another tightly coupled hybrid recommender model with improved accuracy, called Collaborative Deep Learning (CDL). Again, this model uses the bag-of-words to model documents, but the difference is that a stacked denoising autoencoder is used to learn latent item vectors, which are then used in a PMF. An auto-encoder is built up as an hour-glassed shaped neural network to reduce the dimension in the most middle layer as a dense vector and is learnt by reconstructing the same input as output in the last final layer. The goal is to learn
dense representations of items in the most middle layer while keeping the important information of the document. CDL also relies on PMF [11].

Even though pLSA, LDA and Auto-Encoders find semantic relations between words and documents, or topic relations, these methods are based on statistics of word occurrences in documents, which is a broad scope to learn semantics. In order to get better semantics and also polysemy, smaller word windows within documents where the word order is kept must be considered.

3.3 Word-embedding-based recommender systems with extensions

More recently the two unsupervised algorithms GloVe [32] and word2vec [33] have become well renowned for efficient learning of word embeddings/word vectors. Though, the architecture of these two models differ from each other, they share the same goal, which is to cluster words that have similar meanings. Word2vec is a shallow window based method with two modes for learning: either make prediction of a given word based on summation of surrounding words, called continuous-bag-of-words, or make a projection from a given word to the surrounding words, called continuous skip-gram. GloVe is a model that combines global matrix factorization, e.g. pLSA, with local context windows such as n-grams. However, both of these methods are very powerful and Mikolov et al. [33] show that with well clustered word embeddings one can perform algebraic operations on the word vectors, for example: \( \text{vector(“king”) – vector(“man”) + vector(“woman“)} \) has the closest vector representation of the word \textit{queen}. In [20] word vectors were used to create document representations for a content-based filtering recommender system. The technique used was to let the centroid of all word vectors in a document be the item vector representation. The same approach was used to create user profiles, i.e. letting the user vector be the centroid of all item vectors the user liked in the past. The recommendation in [20] is based on the similarity between the user vector and the item vectors by recommending the most similar items. However, one major disadvantage is that with longer documents vector averaging is still having the problem of polysemy, e.g. consider document 1 and document 2 in Table 2.

Two unsupervised extensions of the word2vec algorithm are Skip-thought vectors [34] and Paragraph vectors [35], which are used to create vector representations for sentences and sequences of words respectively. The goal is the same as in word2vec.
but on a higher level, i.e. clustering semantically similar sentences and sequences of words. The Paragraph vector technique is more similar to word2vec and has two different learning modes. Either predict an unordered sequence of words given the paragraph vector, called distributed bag-of-words, or predict a word given the surrounding words and the paragraph vector, called distributed memory model. The paragraph vector technique is used in [36], where item reviews are applied as input to learn paragraph vectors. Item reviews are used to learn Paragraph vectors (i.e. latent vector representations) both for items and users. To predict the rating PMF is used on top to connect the two paragraph vector models. Disadvantages of paragraph vectors are that with larger paragraphs/sequences this method is weakening the use of word order, since it uses vector averaging, and will probably be less efficient at handling polysemy.

The Skip-thought vector [34] method is similar to a machine translation model based on a sequential structure using a type of Recurrent Neural Network, called Gated Recurrent Unit (GRU). It is capable of learning long term dependencies in the sequential input, and encoding a sentence of words into a dense latent sentence vector representation. The encoded sentence vector is then used as input to two GRU decoders to predict the previous and next sentence respectively. In [37] the Skip-thought vector model is pretrained on sentences in item reviews, where the main goal of the Skip-thought model is to learn the encoder to produce semantically similar latent item vectors from similar review texts. These latent vectors are then used in a Factorizing Personalized Markov Chain [38] model, which is a stochastic sequential model combining a type of matrix factorization with Markov Chains. Given a set of purchases it predicts the next most likely purchase. To use Markov Chains requires the knowledge of when an item is purchased. The benefits from using a sequential structure like a GRU to create item latent representations is that word order is taken into account and therefore solve the problem of both synonymy and polysemy.

Benefits with Skip-thought vectors and Paragraph vectors are that if there is little textual data available for the specific task, pretrained word vectors can be used which might improve the results.
3.4 Recommender systems with CNNs

The model proposed by Kim et al. [9] uses a one-layer CNN to extract latent item vector representations from item descriptions and has shown to outperform the earlier hybrid models that are based on the bag-of-word principle in terms of rating prediction accuracy. A CNN uses filters and convolutions to extract important features from the text. The texts are represented by concatenated word vectors to each corresponding document of an item, where each word has its own unique word vector, which lets the word order of each document be intact such that polysemy and synonyms can be learnt. When using a CNN, the word vectors can either be trained from scratch as weights of the CNN, or pretrained with an unsupervised method, such as GloVe or word2vec, and then be finetuned when training the model, which can improve the performance [9]. The technique used for rating predictions is PMF.

Gong et al. [39] proposed an attention based CNN recommender system on text for hashtag recommendation. In this model they treat the hashtag recommendation as a multi-class classification problem where the CNN predicts likely hashtags based on the given text. Seo et al. [40] later used a similar architecture of the CNN to construct a dual CNN system to learn item and user latent feature representations from user review texts. Instead of a multi-class problem they exchanged the final layer to predict ratings by calculating the dot product between the user vector and item vector, which is similar to a matrix factorization approach. Zheng et al. [41] proposed a model that uses two parallel CNNs to create user and item latent vectors from user and item review text respectively. To link the two parallel CNNs together they use Factorization Machine [42], which is a type of factorization model, as the top layer for rating predictions.

A report was recently published by Huan et al. [43] where they introduced Convolutional Denoising Autoencoder (CDA) to extract latent document features. However, the report of Huan et al. [43] is written in Chinese and therefore difficult to understand, but our beliefs are that they follow a similar structure to Wang et al. [11], i.e. perform unsupervised learning on the item latent vectors from the CDA interchangeably with the PMF.

Recommendations can also be based on visual information. Nguyen et al. [44] proposed a personalized CNN based image tag recommender system. It uses the CNN to extract the visual features from images and has a special layer after the
feature extraction where user relevant information, such as tag history, is added. On top of that, a multi-layer perceptron is used to perform multi-class prediction where the scores are used as ranking of tags. Furthermore, Fan et al. [45] proposed a hybrid movie recommender model incorporating visual information in movie trailers instead of text to create latent item vectors. Similar to our work it is a tightly coupled model using a CNN to create latent vectors and PMF as the collaborative model.

The overall benefits from using word vectors and CNNs are that the word order can be taken into account, context in which the words appear can be learnt and different words can be distinguished from one another. To summarize, this means that one can better capture and vectorize the meaning of the document, and two documents with different meaning will most likely end up having different vector representations.

Though, it is not clear whether CNNs or RNNs are better to learn the meaning of a text [46]. The way a CNN is structured, with convolutions and max pooling layers, makes it very efficient as a baseline model to extract important features from text. The work of Kim et al. [9] is the basis to our research questions to see how the sizes of filters, number of filters and dropout rate affect the rating prediction. A more detailed description of the CNN architecture can be found in Appendix A.3.

3.5 Architecture evaluation of CNNs

There has not been much work done in complexity analysis of CNNs in hybrid recommender systems. Kim et al. [10] evaluated how the number of filters affected the rating prediction accuracy, but they did not evaluate different sizes of the filters. Zhang et al. [47] filled the gap of complexity analysis for a one layered CNN in the task of sentence classification. The architecture used in their evaluation is similar to the one we use and covered different sizes of filters, number of filters, pooling strategies, activation functions, and the effect of regularization. Their findings relevant to our work is that filter size and number of filters play a significant role on the performance and 1-max pooling (max-over-time pooling), which we explain in Appendix A, outperformed other pooling strategies.

Furthermore, researchers have experimented with deeper CNN architectures in classification tasks and also using character embeddings instead of word embeddings. Kalchbrenner et al. [48] proposed a deeper CNN structure with a more
complex convolution scheme compared to the one-layer CNN proposed in [49]. The deeper structure is made possible by using smaller one-dimensional filters that span over a window of words instead of bigger two-dimensional filters usually spanning over a window of words and the whole embedding dimension. Conneau et al. [50] proposed an architecture based on the principles of the VGG-network [51] and ResNet [52], which is 29 layers deep. Another difference is that they use character vectors/embeddings instead of word vectors/embeddings. Their proposed model achieve state-of-the-art performance on text classification and sentiment analysis tasks and their beliefs are that deeper networks in NLP should have the same benefits as deeper networks in image processing.

3.6 Pre-training of CNNs and RNNs
Applying pre-trained word embeddings in CNNs has shown to improve the performance in both NLP tasks [49] and in hybrid recommender systems [10]. We have not found any examples of CNNs with unsupervised pretraining on text apart from using pre-trained word embeddings, which LeCun et al. [14] explain is due to the recent advancements in pure supervised training. Unsupervised pre-training is normally done in order to create better initial weights for a feed forward network such that the it can generalize better and thereby preventing the model from overfitting. This is especially needed for smaller data sets with few training examples [14].

Radford et al. [53] recently proposed a Long Short-Term Memory model (LSTM), which is a type of RNN (similar to a GRU) that is capable of learning long-term dependencies, based on UTF-8 encoded bytes in a text. It is pretrained in an unsupervised way, by given a sequence of UTF-8 encoded bytes, it predicts the next possible byte. The model is then used for different classification tasks. Their proposed pretrained model displays state-of-the-art-performance after very few iterations, which also shows that a model can be learnt from much lower level than word vectors. Radford et al. [53] believe that one can possibly improve performance for longer documents with a hierarchical model, i.e. multiple layers.

According to LeCun et al. [14] pretraining is needed for small datasets where it is easy to overfit. They also state that unsupervised training will be more important in the long term. This is motivated by the way humans and animals learn, which is
largely unsupervised, i.e. we discover structures of the world by observing it and are not being told the name of every object.

Since our model is supposed to learn latent item and user vectors from sparse datasets with few ratings, we believe that unsupervised training of the CNN will help create better latent item vectors and improve the rating prediction accuracy.
Chapter 4

Method

In this chapter we present the method which followed a similar setup as Kim et al. [9, 10]. The method includes description of the dataset, cleaning of the datasets, the setup for building the models, the examination method and how we evaluated the result.

4.1 Datasets

Three real world datasets consisting of users’ explicit ratings from 1 to 5 were used to evaluate the models, two from MovieLens and one from Amazon Instant Video (AIV), which were obtained from the website of Kim et al. [54]. The datasets consist of different sparsity to examine the generalisation properties of the algorithms. In the AIV dataset movie reviews written by users are included and used as item description documents. The two datasets from MovieLens did not contain any item description documents, which is why movie plots that Kim et al. [9] collected from IMDB were used as item description documents. Furthermore, since the use of pretrained word embeddings yielded better results than training of word embeddings from scratch [9], the GloVe pretrained word embeddings of dimension 200 is used which is a vocabulary of 400 000 words.

Since all the models have the same input it is expected that the pre-processing of data has a minor effect on different architectures. Statistics of the documents and vocabulary before pre-processing is shown in Table 4 while statistics of the rating data before pre-processing is shown in Table 5.

The same pre-processing as in [9, 10, 11, 12] was used for this project, such that the best performing model proposed in Kim et al. [10] could be used as a starting point without further hyperparameter search. The pre-processing procedure was as follows:
1. The maximum length of item description documents was limited to 300 words. In other words, longer documents were truncated at 300 words.
2. Stop words were removed.
3. TF-IDF was calculated for all words, which is a measure of the importance of words.
4. The top 8000 most distinct words were selected as vocabulary, i.e. the ones with highest TF-IDF score.
5. All the words in the documents that were not contained in the final vocabulary were removed.
6. Words that were not in the pretrained GloVe dataset were initialized with random values.

The items that did not have any description documents were removed from the dataset and users that had less than 3 ratings in the AIV dataset were also removed due to the extreme sparsity of this dataset.

The statistics of the documents and vocabulary and final average length of documents are shown in Table 6 while statistics of the rating dataset is shown in Table 7.

<table>
<thead>
<tr>
<th></th>
<th>Average document length</th>
<th># of pretrained words</th>
<th># of words in vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MovieLens-1m</strong></td>
<td>194 words</td>
<td>23 257</td>
<td>26 210</td>
</tr>
<tr>
<td><strong>MovieLens-10m</strong></td>
<td>180 words</td>
<td>36 429</td>
<td>43 481</td>
</tr>
<tr>
<td><strong>AIV</strong></td>
<td>1 257 words</td>
<td>85 279</td>
<td>162 833</td>
</tr>
</tbody>
</table>

Table 4. Statistics of the documents before pre-processing as average document length in number of words, number of pretrained words in the vocabulary and total number of words in the vocabulary.

<table>
<thead>
<tr>
<th></th>
<th># users</th>
<th># items</th>
<th># ratings</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MovieLens-1m</strong></td>
<td>6 040</td>
<td>3 544</td>
<td>993 482</td>
<td>4.641%</td>
</tr>
<tr>
<td><strong>MovieLens-10m</strong></td>
<td>69 878</td>
<td>10 073</td>
<td>9 945 875</td>
<td>1.413%</td>
</tr>
</tbody>
</table>
Table 5. Statistics of the three rating datasets before pre-processing for number of users, items and ratings. Density is calculated by dividing the number of ratings with the product of number of items and number of users.

<table>
<thead>
<tr>
<th></th>
<th># users</th>
<th># items</th>
<th># ratings</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens-1m</td>
<td>6 040</td>
<td>3 544</td>
<td>993 482</td>
<td>4.641%</td>
</tr>
<tr>
<td>MovieLens-10m</td>
<td>69 878</td>
<td>10 073</td>
<td>9 945 875</td>
<td>1.413%</td>
</tr>
<tr>
<td>AIV</td>
<td>29 757</td>
<td>15 149</td>
<td>135 188</td>
<td>0.030%</td>
</tr>
</tbody>
</table>

Table 6. Statistics of the documents after pre-processing as average document length in number of words, number of pretrained words in the vocabulary and total number of words in the vocabulary.

<table>
<thead>
<tr>
<th></th>
<th>Average document length</th>
<th># of pretrained words</th>
<th># of words in vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens-1m</td>
<td>97 words</td>
<td>7 927</td>
<td>8000</td>
</tr>
<tr>
<td>MovieLens-10m</td>
<td>92 words</td>
<td>7 972</td>
<td>8000</td>
</tr>
<tr>
<td>AIV</td>
<td>92 words</td>
<td>7 980</td>
<td>8000</td>
</tr>
</tbody>
</table>

Table 7. Statistics of the three rating datasets after pre-processing for number of users, items and ratings. Density is calculated by dividing the number of ratings with the product of number of items and number of users.

4.2 Model
We used the same model architecture as Kim et al. [10], but the algorithms were reimplemented in two versions due to memory and compatibility problems. When we started this project, we were working on a laptop windows machine where TensorFlow was only compatible with Python 3 versions. On top of that Keras 2.0.8 was used with CUDA 6.0 GPU library on an NVIDIA GeForce GTX 760M graphic...
card. As we ran initial tests we realised that this setup could only handle smaller datasets and models due to memory limits in the GPU.

To solve the problem with memory limits we used the Machine Learning engine on the Google Cloud platform. However, on the Google Cloud platform TensorFlow was only compatible with Python 2.7, which is why we reimplemented the code to also fit this setup. On top of that Keras 2.0.8 was used with Tesla K80 GPUs for computations.

As debugging was more convenient on the laptop we decided to keep the first version for debugging and pre-processing of data while the second Google Cloud version was used to run all the experiments. The two versions of the code were nearly identical apart from some version specific changes in method names and functions, which were carefully tested on smaller datasets to deliver the same results.

The model consists of a CNN which is tightly coupled with PMF. A detailed explanation of the PMF can be found in Appendix A1, the unification of the two models is explained in Appendix A.2 while a detailed explanation of the CNN architecture is found in Appendix A.3. The optimization methodology of the unified model is found in Appendix A.5. For pretraining we implemented a Convolutional Autoencoder which we explain in more detail in Appendix A.4.

The hyperparameters that were not manipulated in our simulations were set to the values reported in [10]. These were:

- The latent vector dimension which was set to 50. The dimension of this vector determines the complexity of the PMF model. For example: a smaller dimension makes it more prone to underfit objective function of predicting ratings while a larger dimension makes it more prone to overfit.

- The regularization terms $\lambda_u$ and $\lambda_v$, which were set to the values shown in Table 8. These regularization terms are used in the PMF for the updates of latent user and item vectors respectively. For example: a higher value means that the resulting weight update is punished more, i.e. the resulting update per iteration is smaller. Kim et al. [10] argue that a high $\lambda_v$ makes update changes smaller from the PMF such that it can learn more from the CNN.
• In the CNN we used minibatches of size 128 with Root Mean Square propagation (RMSprop), which adapts its learning rate for each weight, as optimization function and mean squared error as the objective function.

The convolutional autoencoder was used to pretrain all layers in the CNN.

Autoencoders are common unsupervised models used for dimensionality reduction, which learn dense latent representations of the input data. Autoencoders are also commonly used for the pretraining of deep neural network models [55]. We set the encoding network of the convolutional autoencoder to have the same architecture as the original model while we implemented the decoding network of the convolutional autoencoder as a mirrored version of the encoder network. The final layer of the decoding network was set to predict the words that were used as input to the encoding network. A more detailed explanation can be found in Appendix A.4. The setup for the convolutional autoencoder was:

• A batch size of 8, due to memory limits, with RMSprop as optimization function and binary cross entropy as the objective function. Binary cross entropy was used since it is more suitable for classification problems, such as predicting words.

<table>
<thead>
<tr>
<th></th>
<th>AIV</th>
<th>MovieLens-10m</th>
<th>MovieLens-1m</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_U$</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$\lambda_W$</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 8. The values of the regularization terms for the PMF that are used in our model.

4.3 Examination method

The complexity of the model was examined by applying grid search in the hyperparameter space, consisting of the filter size, number of filters and dropout rate, for all three datasets. This resulted in 72 models per dataset. The following values per hyperparameter were chosen:

• To see the importance of filter size, one filter size was chosen per model where filter sizes used were 1x200, 3x200 and 5x200. The numbers 1, 3 and 5 determine the number of words they cover while 200 is the length of the word
embedding. Further, we refer to the filters as the number of words they cover, i.e. filter size of 1, 3 and 5.

- To see the importance of number of filters, 75, 150, 300 and 600 filters were used.
- To see the importance of dropout rate, 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0 were used. Since a dropout rate of 1.0 is not allowed in TensorFlow, a dropout rate of 0.9999999999 was used instead. This was an arbitrary chosen value which we assumed to be sufficiently near 1.0. The result of having 1.0 as a dropout rate is that all weight updates are turned to 0, which means that the weights do not change during training. The motivation for this was to simulate a model which only relies on PMF, i.e. not using additional information from the CNN on the dataset, without rewriting the code. In the remaining part of the report, this is referred to as not using additional information.

As explained earlier, this setup resulted in 72 models per dataset. Since training many models is a time-consuming and expensive task, we decided to answer the second question of this thesis by only applying the pre-training on 12 models per dataset, i.e. one model per filter size and number of filters. By choosing one model per filter size and number of filters we can see if pretraining has a different effect on smaller or larger filter sizes and on the number of filters. The selected dropout rate per model was selected from the respective non-pretrained best performing model on the validation set. Unfortunately, no statistical comparisons had been made at this point, which means that the selection of “best performing” models was not well grounded and only best on fragmentary evidence – limited observations and trends.

### 4.4 Evaluation

To evaluate the performance of the models the datasets were randomly split into 80% training data, 10% validation data and 10% test data. The dataset split was performed once per dataset, which means that the same training, validation and test subsets were used for all models.

The evaluation metric used was the Root Mean Square Error (RMSE) on the rating prediction accuracy. The RMSE was chosen since it is often used as a performance metric in collaborative filtering for rating prediction and is closely related to the
objective function exploited in the network weight optimization. The RMSE was calculated in the following way:

\[
RMSE = \sqrt{\frac{\sum_{i,j}^N M (r_{ij} - \hat{r}_{ij})^2}{\# \text{ratings}}},
\]

where the \( r \) is the true rating, \( \hat{r} \) is the predicted rating, \( N \) is the total number of items and \( i \) is indexing items, \( M \) is the total number of users and \( j \) is indexing users while \( \# \text{ratings} \) is the total number of ratings in the dataset.

Each model was trained for 200 iterations at most where training of the model stopped when the RMSE of the validation data had been increasing for 20 (endure count) iterations in total (form of early stopping). This prevented the model from overfitting. The reported result is the RMSE on the test data from the model which had the lowest RMSE on the validation data.

Due to the heavy computations only one model per each combination of the filter size, the number of filters and dropout rate were evaluated (72 models per dataset). For reliability, the average performance and the standard deviation were calculated:

i) over the multiple numbers of filters to show trends for filter size and dropout, i.e. marginalisation of the results over the number of filters.

ii) and over the top 3 dropout rates, except dropout rate 1.0, to show trends for filter size and number of filters, i.e. marginalisation of the top 3 results over the dropout rate. The reason we excluded dropout rate 1.0 is because we wanted to see the effect of using the different filter sizes and number of filters in the CNN. When using a dropout rate of 1.0 all updates of weights become 0 such that the CNN becomes untrainable. This results in that the model only uses the PMF to predict ratings.
Chapter 5

Results

In this section the performance of the model is presented. Since we only had one result per hyperparameter configuration with the goal to see trends in the results, we chose to calculate the average performance of different values of the filter size and dropout rate in the following way:

\[ m_{i,j} = \frac{\sum_k x_{i,j,k}}{N}, \]

where \( x \) is the RMSE, \( i \) is the filter size (ranging over \{1, 3, 5\}), \( j \) is the dropout rate (ranging over \{0.0, 0.2, 0.4, 0.6, 0.8\}), and \( k \) is the number of filters (ranging over \{75, 150, 300, 600\}). In other words, we performed marginalisation of the results over the number of filters.

The average performance per filter size and number of filters was chosen to be calculated from the top 3 results per filter size and the number of filters, as reported from the validation set. This choice was motivated by our curiosity in knowing trends for filter size around the best results that our models delivered. The average performance per filter size and the number of filters was calculated in the following way:

\[ m_{i,k} = \frac{\sum_T \min (y_{i,k})}{T}, \]

where \( y \) is the RMSE, \( i \) is the filter size (ranging over \{1, 3, 5\}), \( k \) is the number of filters (ranging over the \{75, 150, 300, 600\}), while \( t \) is the ranging from 1 to \( T \). In our case \( T \) is 3 since we chose the three lowest values, \( \min \) can be seen as a “draw minimum without replacement”-function and \( : \) is a notation of choosing the minimal RMSE from the set of different dropouts. In other words, we performed marginalisation of the top 3 results over the dropout.
5.1 AIV results

In general, the result on the AIV which is the sparsest dataset, had greater deviations compared to the two denser datasets, i.e. the spread between the lowest RMSE and the highest is greater than the spread for the MovieLens-10m and MovieLens-1m datasets. This was especially noticed for different dropout rates and filter sizes.

The general effect of different filter sizes on the AIV dataset was that smaller filters tend to give better results. There is also a tendency that smaller filters make the model less sensitive to dropout, i.e. the difference between the highest and the lowest RMSE is smaller for smaller filters, which can be seen in Figure 4. What also can be seen in Figure 4 are trends of the enhanced performance with a dropout rate of 0.8 for filter sizes 3 and 5, while filter size 1 has enhanced performance with a dropout rate of 0.6.

The trends for the number of filters is less distinct than those for the filter size and dropout rate. Figure 5, which shows the average performance per filter size and the number of filters, suggests improved performance for the lower number of filters and smaller filters.
Figure 4. The average performance per filter size and dropout for the AIV dataset, which is the sparsest dataset. Since we only had 1 result per configuration, the average is calculated from a set where number of filters is ranging over 75, 150, 300 and 600 filters. The standard deviation, multiplied with 0.5, is represented by the bars at each data point.
Figure 5. The average performance per filter size and number of filters for the AIV dataset. Since we only had 1 result per hyperparameter configuration the average in this figure was calculated from the top 3 results per filter size and number of filters. The standard deviation, multiplied with 0.5, is represented by the bars at each data point.

5.2 MovieLens-10m

The results of MovieLens-10m data set, which is the second sparsest, show trends of better performance when the dropout rate is around 0.6-0.8. What we also see is that the spread between the lowest RMSE and the highest is smaller than the spread for the AIV dataset but bigger than the spread for the MovieLens-1m dataset.

The general effect of the size of filters is, similar to the AIV dataset, that the performance tended to be better for smaller filter size 1 with a dropout rate of 0.6-0.8 while filter size 3 and 5 showed enhanced performance for dropout rate 0.8. This can be seen in Figure 6 where the average performance per filter size and dropout is shown. However, different from the AIV dataset is that smaller filters do not seem to make the model less sensitive to dropout. The observed trend suggest that the sensitivity of dropout is similar regardless of filter size.
Regarding the effect of the number of filters there is a more distinct trend of better performance for the lower number of filters across all the tested filter sizes. This can be seen in Figure 7, which shows the average performance per filter size and the number of filters. Another difference from the AIV dataset is that the performance is similar for all three filter sizes when using less number of filters.

Figure 6. The average performance per filter size and dropout for the MovieLens-10m dataset, which is the second sparsest. Since we only had 1 result per configuration, the average is calculated from a set where number of filters is ranging over 75, 150, 300 and 600 filters. The standard deviation, multiplied with 0.5, is represented by the bars at each data point.
The average performance per filter size and number of filters for the MovieLens-10m dataset. Since we only had 1 result per hyperparameter configuration the average in this figure was calculated from the top 3 results per filter size and number of filters. The standard deviation, multiplied with 0.5, is represented by the bars at each data point.

5.3 MovieLens-1m

Comparing MovieLens-1m, which is the densest dataset, with the two sparser datasets one can see that the performance tends to be less sensitive to dropout, i.e. a smaller difference between the lowest and the highest RMSE, regardless of what filter size is used. The results also show trends, that all models regardless of filter size and number of filters are improving the performance if additional information is used, i.e. if the CNN has a dropout rate equal to 0.8 or less. What we also see is that the difference between the lowest and the highest RMSE is smaller than the difference for both the AIV and MovieLens-1m dataset.

The general effect noticed of the filter size is, similar to the AIV and MovieLens-10m, that the performance tended to be better for smaller filters, and with a dropout rate of 0.2, which can be seen in Figure 8 where the average performance per filter size
and dropout is shown. The trend for filter size 3 and 5 is less distinct but suggests enhanced performance with a dropout of 0.4-0.6.

Regarding the effect of the number of filters there is a slightly different trend compared to the sparser datasets. In Figure 9, which shows the average performance per filter size and number of filters, one can see that the trend for filter size 1 was a better performance with higher number of filters while the trend for filter size 3 and 5 was a better performance with a lower number of filters.

![Figure 8](image)

Figure 8. The average performance per filter size and dropout for MovieLens-1m, which is the densest dataset. Since we only had 1 result per configuration, the average is calculated from a set where number of filters is ranging over 75, 150, 300 and 600 filters. The standard deviation, multiplied with 0.5, is represented by the bars at each data point.
37

Figure 9. The average performance per filter size and number of filters for the MovieLens-1m dataset. Since we only had 1 result per hyperparameter configuration the average in this figure was calculated from the top 3 results per filter size and number of filters. The standard deviation, multiplied with 0.5, is represented by the bars at each data point.

5.4 Result of pretraining
In this section we present the evaluation of pretraining in comparison to non-pretrained CNNs. Due to limited resources, we chose to only pretrain models with the same configuration as the best performing models reported on the validation set. These models were chosen per filter size and the number of filters for different dropout rates. In other words, for each filter size and number of filters, we chose the same dropout rate as the model with the lowest RMSE, as reported on the validation set. The reason we chose these hyperparameter setups was due to our beliefs that filter size and number of filters could have a bigger effect on the pretrained CNN than dropout rate. The formula used to find the lowest RMSE was:

\[ m_{i,k} = \min (z_{i,j,k}) \]
where \( z \) is the RMSE, \( i \) is the filter size (ranging over \( \{1, 3, 5\} \)), \( k \) is the number of filters (ranging over the \( \{75, 150, 300, 600\} \)) and \( : \) is a notation of choosing the minimal RMSE from the set of different dropouts. Since the result of pretrained CNNs and non-pretrained CNNs were very similar we chose to present the results as the average over all parameters with the corresponding standard deviations.

The results can be seen in Table 9, which shows the average performance per dataset with its corresponding standard deviation. As we can see, the results for both MovieLens datasets are almost identical, while there is a slight difference for the AIV dataset. For the AIV dataset the non-pretrained CNN had both a lower average RMSE and lower standard deviation while the pretrained CNN had a slightly higher average RMSE and higher standard deviation. This means that there are tendencies of better overall performance of the non-pretrained models. However, the results are based on limited observations and are not tested for statistical significance.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Non-pretrained</th>
<th>Pretrained</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AIV</strong></td>
<td>1.104 (0.009)</td>
<td>1.109 (0.012)</td>
</tr>
<tr>
<td>MovieLens-10m</td>
<td>0.785 (0.001)</td>
<td>0.785 (0.001)</td>
</tr>
<tr>
<td>MovieLens-1m</td>
<td>0.847 (0.002)</td>
<td>0.847 (0.003)</td>
</tr>
</tbody>
</table>

Table 9. The average performance and standard deviation of the chosen results as described in the beginning of section 5, and the respective average performance of the pretrained models. The results show that the performance is very similar for both MovieLens datasets while there is a slight difference for the AIV dataset.
Chapter 6

Discussion

The goal of this paper was to complement earlier research in recommender systems which make use of CNNs in a tightly coupled model with a PMF. To do so we evaluated the complexity of the CNN, more specifically the filter size, number of filters and the dropout rate and how they affect the performance of the model as a whole. In earlier research on CNNs in combination with PMF there has been evaluations on the CNN with different number of filters but not with different filter sizes and evaluation on the effect of different dropout rates. In order to focus on these three hyperparameters we decided to fix all other hyperparameters, e.g. the depth of the CNN in terms of layers, the dimension of the output vector of the CNN and the hyperparameters for the PMF model. All the hyperparameters that we kept fixed were chosen from the best praxis in earlier research.

With that said, there are many other hyperparameters that could affect the performance, two of which are closely related to the update of weights in the CNN: the weighting of the regularization in the PMF, $\lambda_U$, for user latent vectors and, $\lambda_V$, for item latent vectors. In the work of Kim et al. [10] they investigated the best performing values of the regularization terms in the PMF, however this was for a PMF tightly coupled with a CNN using a dropout of 0.8 (0.2 if the model is built in the Theano framework as backend instead of TensorFlow), a combination of filter sizes 3, 4 and 5 and 300 filters in total. Since we used a range of different hyperparameter values in the CNN, retuning of the regularization weighting terms, $\lambda_U$ and $\lambda_V$, per configuration could affect the performance.

Regarding the dropout rates in our results, the trends for filter size 3 and 5 demonstrated better performance with a higher dropout rate compared to using filter size 1. This is most likely due to the fact that the PMF, in earlier research, was configured in combination with a CNN that had a dropout of 0.8 and a combination of filter sizes 3, 4 and 5. We also observed that the less complex models, in terms of smaller filters and less number of filters, exhibited trends of better performance with
a smaller dropout rate compared to the models with more and larger filters. The explanation could be that less complex models are less prone to overfit to training data compared to a more complex model, and thus not require a high dropout in order to minimise the risk for overfitting. Since our results show trends of better performance of the filter size of 1, we suggest that the CNN finds keywords to be more useful instead of a sequence of words.

For the majority of the results we also observed trends of better performance for less number of filters, i.e. 75 filters. Since it was still an improving trend at 75 filters for most results one could argue that even less number of filters could give better performance. This suggests that the model finds either less number of words important or that many similar words gets triggered by the same filters. However, more experiments are required to support this hypothesis.

The results obtained using the pretrained network show no trends of improvement compared to the non-pretrained CNN on the two MovieLens datasets, where it had slightly worse performance compared to the non-pretrained CNN on the AIV dataset. The reasons for this could be that the input data was not as complex as we initially believed or that the CNN had good abilities to learn with pure supervised training such that no pretraining was required. Another reason could be that the pretraining was not well performed, i.e. the autoencoder did not improve the initial weights in the CNN which could be due to bad hyperparameter setup. The reason we suspect this is due to a high accuracy on the validation set after a few iterations which is why further investigations are needed to tell the effect of pretraining with this model architecture.

As described in chapter 2 and chapter 3 there has been a lot of research in creating better latent vectors for words and text in general. Some of those take word order into account such that the problem of synonyms and polysemy is handled, i.e. find similarities between text with different words but same meaning and distinguish between texts with similar words but different meanings. However, our experimental results showed trends in increased performance for filter size 1. The difference between using a filter size of 1 and larger filters is that a filter size of 1 finds information from singular words in the item description while larger filters extract information from a sequence of words. This means that word order is not taken into account and that state of the art performance can be achieved by only using
contextual similarities of words as our best results had an average RMSE of 1.097 for AIV, 0.784 for MovieLens-10M and 0.845 for MovieLens-1M, compared to Kim et al. [10] who had the best average RMSE of 1.101 for AIV, 0.784 for MovieLens-10M and 0.847 for MovieLens-1M. This would suggest that the word ordering was not as important as we initially thought.

Therefore, one could also experiment with even smaller filters, such as Conneau et al. [50], who achieved state-of-the-art performance on different classification with smaller filters and deeper structure. This would suggest that the CNN does not need to learn language the same way humans do, i.e. instead of reading words it could find its own translation of texts.

Furthermore, the experimental results of our models which did not make use of the CNN, i.e. had a dropout of 100%, performed only a few percent worse than state-of-the-art. In the report of Kim et al. [10] their pure PMF model achieved an average RMSE of 1.412, 0.831 and 0.897 on the AIV, MovieLens-10m and MovieLens-1m respectively. Our results on the PMF were 1.129, 0.794 and 0.880 on the AIV, MovieLens-10m and MovieLens-1m respectively. The difference was that our PMF model also considered statistics for the rated items, in simple terms weighting the latent item vectors importance’s with respect to how many ratings they had. This also speaks for that simple techniques can come close to the state-of-the-art performance. However, since we only had one dataset per AIV, MovieLens-10m and MovieLens-1m datasets, more experiments are required to support this hypothesis.

6.1 Limitations

One of the main drawbacks of this report was that we did not provide any statistical significance or hypothesis testing for our results. The resources were limited, and computations were too demanding to be run multiple times per configuration, which was why only one split for training, validation and test set was made for each dataset. This resulted in inconclusive outcomes that can only rest on the observed trends for the different configurations on the three datasets.

Due to time limits the pretrained CNN was not thoroughly explored. The pretraining was performed using the pretrained word vectors as initial weights. To not overfit to the training data early stopping was used after the performance of the validation set had decreased for a number of iterations. A qualitative analysis of the Convolutional
Autoencoder would have been necessary to see the effect of pretraining, e.g. compare the word vectors before and after the pretraining. Furthermore, it would have been useful to see if the latent vectors or filters had learnt any useful information.

6.2 Ethics

Most datasets used for recommender systems contain private data. In our case we used data that has been collected from users’ interactions with products. The private information, such as user names, has been anonymized by random identification numbers but even though private information has been anonymized there is still a risk of de-anonymization, or reverse engineering, to connect data back to its real user. This is a known risk, but without publicly available datasets the research on recommender systems and other areas would stall.

Another aspect is the actual recommendations that are given to users. While recommender systems make it a lot easier to find suitable products in large selections, they can also feed people’s addiction. For instance, making people spend more time and money on a product than they can afford, e.g. watching movies, playing games or buying products they do not need.

Since basically everything can be recommended, given history of a user’s actions, there are other ethical aspects one has to be aware of. A user will most likely use, e.g. the internet, for personal interests and beliefs which will result in recommendations built on the user’s actions. This is an example of a filter bubble where no new information is recommended, only recommendations that confirm the user’s initial beliefs. An extreme example is fake news pages. The more a user read fake news, the more a user get recommended fake news, and the stronger they will believe this news are true.
Chapter 7

Summary and conclusion

Here we present the major conclusions and best praxises for implementations of similar models that are based on the experimental results. The experiments were based on three rating datasets with different sparsity and two item description datasets where each rating dataset and its corresponding item description dataset were split into training, validation and test subsets once. As a result, we showed trends from our experimental results where similar performance would be expected for similar datasets.

7.1 Summary

Regarding the dropout, there were trends of less dropout for a less complex model. An explanation could be that smaller filters and less number of filters are less prone to overfit to training data such that dropout is not needed to the same extent. Furthermore, the model tended to be more sensitive to change in hyperparameters on sparser datasets.

The results showed trends of better performance with smaller filters. This might be due to that keywords are more important than a sequence of words. Furthermore, the majority of our results showed trends of better performance with lower number of filters. Some also had an improving trend with 75 filters which means that less than 75 filters could improve the performance even more. One could also argue that maybe even smaller filters, such as in Conneau et al. [50], could improve the performance. However, in that case it is needed to leave the theory of having filter sizes of the same width as the whole word embedding dimension.

As for the pretraining of the CNN the results show no improvement. This could be due to that the model is fairly simple, not very deep, and has good abilities to learn with pure supervised training.
7.2 Further work

For further work we have a few questions that could be interesting to look into.

- Since our results showed trends in better performance with the lower number of filters and filter sizes it could be valuable to investigate the lower range further.

- In our experiments we had the regularization terms of the PMF, $\lambda_u$ and $\lambda_v$, fixed. How will change of these regularization terms affect the performance and the dropout rate of the CNN?

- To get a better understanding of the CNN and its usefulness in the recommender system one could investigate the qualitative performance of the CNN, e.g. investigate the learnt word vectors, the most important words in a document or the learnt latent item vectors.

- Since the structure we used for the Convolutional Autoencoder to pretrain our CNN was never tested before we did not know the effect of the pretraining with this structure. A qualitative analysis for different architecture of Convolutional Autoencoders could be interesting to see their learning abilities.

- Even simpler model improvements of the PMF could also be interesting to evaluate and compare to the more advanced recommender models.
References


Appendix

A.1 Probabilistic Matrix Factorization

The goal of Matrix Factorization (MF) is to find latent vectors for users and items and let the inner product of these approximate the value of the rating for the given user and the specific item. Suppose that we have N users and M items, and a user rating matrix \( R \in \mathbb{R}^{N \times M} \). In MF, the model is built up by having a k-dimensional vector \( u_i \in \mathbb{R}^k \) per user and a k-dimensional vector \( v_j \in \mathbb{R}^k \) per item such that \( U \in \mathbb{R}^{k \times N} \) and \( V \in \mathbb{R}^{k \times M} \). These matrices will represent the latent features for the users and items respectively. The rating \( r_{ij} \) of user \( i \) on item \( j \) will be approximated by the inner product of the two latent feature vectors of user \( i \) and item \( j \), such that \( r_{ij} \approx \hat{r}_{ij} = u_i^T v_j \) which is the same as \( R \approx \hat{R} = U^T V \). The usual way to train this model is by minimizing the sum-of-squared-errors as a loss function, i.e. the squared difference between the actual rating and the predicted rating.

To avoid overfitting, it is common to use regularization. There are different regularization techniques where the most common one is the L2 norm. For Probabilistic Matrix Factorization (PMF) \[30\] with Maximum a Posteriori (MAP) as optimization methodology the regularization terms are equivalent to L2 norms \[30\] and the simplified equation used is:

\[
L = \sum_i^N \sum_j^M I_{ij} (r_{ij} - u_i^T v_j)^2 + \lambda_u \sum_i^N \|u_i\|^2 + \lambda_v \sum_j^M \|v_j\|^2,
\]

where \( I_{ij} \) is an indicator function which returns 1 if user \( i \) rated item \( j \) and 0 otherwise.

A.2 Probabilistic Matrix Factorization and CNN

To understand the optimization methodology, we need to consider the whole model from a probabilistic point of view and how it differs from a standard PMF. In our model we follow the same procedure as \[9, 10, 11, 12\]. At the top level we have the conditional distribution over observed ratings that are given by:

\[
p(R|U,V,\sigma^2) = \prod_i^N \prod_j^M N(r_{ij}|u_i^T v_j, \sigma^2)^{I_{ij}},
\]
where \( N(x|\mu, \sigma^2) \) is the probability density function of the Gaussian normal distribution with mean \( \mu \), variance \( \sigma^2 \) and \( I_{ij} \) is the indicator function that is equal to 1 if user \( i \) rated item \( j \) and 0 otherwise.

For the latent user vector there is no difference from a standard PMF. Each latent user vector will be drawn randomly from a zero-mean normal distribution with variance \( \sigma_u^2 \). The matrix of drawn user latent vectors is given by:

\[
p(U|\sigma_u^2) = \prod_i^N N(u_i|0, \sigma_u^2 I).
\]

The difference comes for the latent item vector \( v_j \) which, as explained in [9], is generated from three variables:

1. The internal weights \( W \) of the CNN that are drawn from a zero-mean normal distribution \( p(W|\sigma_W^2) = \prod_k N(W_k|0, \sigma_W^2) \)
2. The item document descriptions \( X \), which consist of all document descriptions
3. An item specific noise vector \( \epsilon_j \) that is drawn from a zero-mean normal distribution \( p(\epsilon|\sigma_\epsilon^2) = \prod_j^M N(\epsilon_j|0, \sigma_\epsilon^2 I) \)

We denote the latent document vector extracted from the CNN as \( \text{cnn}(W, X_j) \) and let the final item latent vector \( v_j \) be an addition between the extracted latent document vector and the noise vector:

\[
v_j = \text{cnn}(W, X_j) + \epsilon_j.
\]

The conditional distribution of the item latent vector is then given by:

\[
p(V|W, X, \sigma_\epsilon^2) = \prod_j^M N(v_j|\text{cnn}(W, X_j), \sigma_\epsilon^2 I),
\]

where \( \text{cnn}(W, X_j) \) is the latent document vector extracted from the CNN and used as the mean value, and the noise vector \( \epsilon_j \) is used as the variance. This will make the evaluation of the CNN into a squared error function between the item latent vector \( v_j \) from the previous iteration in the training process and the new \( \text{cnn}(W, X_j) \), in other words treating the document latent vector from the previous iteration as true values during training.
A.3 One layered convolutional neural network for text

The CNN model proposed by Kim et al. [10], which is the foundation of our model, is based on the work of [49, 56]. The model is based on the assumption that convolutions extract n-gram features from the text and therefore uses different sizes of n-grams (window sizes) in order to model short and long span relations from the text.

The objective of the Convolutional Neural network is to generate latent document features from the text that describes the items. The model proposed by [10] is consisting of four layers:

1. Embedding layer
2. Convolutional layer
3. Pooling layer
4. Output layer

A toy example of the CNN architecture is visualised in figure 1, and will be used to explain the model in more detail.
The embedding layer is mapping the words from the document into their specific word embeddings which result in a matrix representation, described as document matrix in Figure 1, that will be used for the following convolutional layer. The order of words will be maintained, and the matrix will have dimensions $l \times p$ where $p$ is the size of the word embedding dimension for each word and $l$ is the length of the item document. In figure 1, the text “Highly recommend this show to anyone” is an example of an item document where the length $l$ is 6 and the embedding dimension $p$ is 5.
**Convolutional layer**

The purpose of the convolutional layer is to extract meaningful contextual features from the text. A convolution is a mathematical operation between the input matrix and a filter that is swept over the input matrix with a specific stride, i.e. how many rows and columns the filter will be shifted per convolution. In this architecture the width of the filter has the same dimension as the word embedding size $p$, which is why the stride is only moving in the row dimension. The height of the filter, number of rows, is often called the window size $ws$ is determining how many words that should be considered in the convolution. The convolved value $c_i$ which is the $i$:th element of the feature map showed in figure 1, is calculated in the following way:

- an element wise multiplication between the filter and the chunk of word embeddings.
- summation of the element wise multiplications, which results in a scalar.
- a bias $b_i$ is added to the scalar value.
- and at last, pass the calculated value through a nonlinear transformation function.

The mathematical expression for the convolved contextual feature is

$$c_i = f(W \ast D_{(:,i:ws-1)}) + b_i$$

Where $W$ is the filter, $D$ is the document matrix, $\ast$ denotes the convolution operation and $f$ denotes the activation function. The filter is applied to each possible window of words in the input document \{D(:,1:ws), D(:,2:ws+1), ..., D(:,l-ws+1)} which produces the contextual feature vector (the feature map)

$$C = [c_1, c_2, ..., c_{l-ws+1}]$$

The size of the feature map will depend on the size of the filter and the stride. With a stride of one, the feature map becomes a one-dimensional vector of length $l-ws+1$ as illustrated in figure 1 and the number of feature maps is equal to the number of different filters.

As for the nonlinear transformation function there are many variations that can be used where examples are sigmoid, tanh and rectified linear unit (ReLU). ReLU is a
simple but effective nonlinear function which cuts any value below 0 to become 0 and any value above 0 to be linearly transformed. In our architecture ReLU is used since it typically causes faster learning and better performance [57].

**Pooling layer**

The pooling layer will extract the most important features from the feature maps. Since the feature maps will be of variable length, a max-over-time-pooling (1-max-pooling) is performed. One could also do average pooling but since the most distinctive features are wanted max pooling is considered the better choice to extract those features. As shown in figure 1, the output from the pooling layer is a vector of same length as the number of filters used in the convolutional layer.

**Output layer**

Normally for classification tasks this layer is projected into classes. In our case, as can be seen in figure 1, the projection is to a k-dimensional latent feature vector which is used for the recommendation task. A fully connected layer with dropout between the pooling layer and the output layer is used and the output is the document latent vector with the following function:

\[ s = \tanh(W \{ \tanh(Wd + b) \} + b) \]

As regularization, drop out is used between these layers since it has shown to be a very effective regularization technique [57]. Drop out means that during training weights between the neurons in the fully connected layer has a probability of becoming dead, i.e. get the weight value of zero.

To simplify further model specific explanations, \( CNN(W, X_j) \) will be used to express the document latent vector produced by the CNN, where \( j \) is representing the \( j \):th document in the data set, \( X_j \) is its textual input and \( W \) are the internal weights.

**A.4 Convolutional autoencoder**

An autoencoder can be seen as one network put together with two components, an encoder part and a decoder part. The encoder is supposed to create a dense latent feature vector from the input while the decoder is supposed to recreate the input from the dense latent feature vector. In our convolutional autoencoder we use the
same network as explained in Appendix A.3 as the encoder, while this section explains the decoder.

The decoder was set up with the following structure, which also can be seen in Figure 2:

1. Latent item vectors as input
2. Two fully connected layers
3. Two deconvolutional layers
4. Output layer.

**Fully connected layers**

The two fully connected layers project the dense latent item vector onto a larger dimension. In our case, the output from each fully connected layer was the same as the corresponding number of filters for the encoder.

**Deconvolutional layers**

A deconvolutional layer, or a convolutional transpose layer, is supposed to be the inverse of a convolutional layer. Instead of finding a few important features from a lot of data it is supposed to recreate a lot of data from a few features.

The first deconvolutional layer projected the output from the fully connected layer into the same length as the maximum number of words in a document which is 300. The number of filter used was the same as in the encoder such that the output became $300 \times \text{number of filters}$.

The task of the second deconvolutional layer was to project the output into the same dimension as the document matrix. To do this we used filter sizes of the same length as the filter size in the encoder but the width of 1. The number of filters used were the same number as the word embedding dimension, i.e. 200. The resulting output dimensions was the same as the document matrix, i.e. $300 \times 200$.

**Output layer**

The output layer consisted of a time distributed prediction layer. The reason we chose this type of prediction was due to its ability to keep the order of words, i.e. each row of the document matrix predicts to each corresponding word as a one hot representation where a simplified example is shown to the right in Figure 2.
Figure 2. Example architecture of the decoder in the autoencoder. The size of the filters in the second deconvolutional layer varied between 1, 3 and 5.
A.5 Optimization methodology

We use the same optimization methodology as [9, 10, 11, 12] to optimize the variables in the model, which is based on Maximum a Posteriori (MAP) estimation. The goal is to maximize the probability for user latent vectors, item latent vectors and weights in the CNN, given their respective distribution, ratings, document descriptions, which is shown in the following equation:

$$\max_{U,V,W} p(U,V,W | R,X,\sigma^2,\sigma_U^2,\sigma_V^2,\sigma_W^2)$$

$$= \max_{U,V,W} [p(R | U,V,\sigma^2) p(U | \sigma_U^2) p(V | W,X,\sigma_V^2) p(W | \sigma_W^2)]$$

By taking the negative logarithm of the MAP estimate the following loss function is obtained.

$$\mathcal{L} = \sum_{i}^{N} \sum_{j}^{M} \frac{1}{2} (r_{ij} - u_i^T v_j)^2 + \frac{\lambda_U}{2} \sum_{i}^{N} \|u_i\|^2 + \frac{\lambda_V}{2} \sum_{j}^{M} \|v_j - \text{CNN}(W,X_j)\|^2$$

$$+ \frac{\lambda_W}{2} \sum_{k}^{W_k} \|W_k\|^2$$

Where $\lambda_U$ is $\sigma^2/\sigma_U^2$, $\lambda_V$ is $\sigma^2/\sigma_V^2$ and $\lambda_W$ is $\sigma^2/\sigma_W^2$. $\lambda_U$ and $\lambda_V$ are weights for the regularizing terms that are model and data dependent which need to be manually tuned. Here we have two square loss functions, first one is between the true rating and the predicted rating while the second one is between the item latent vector from the previous iteration and the predicted item latent vector from the current iteration.

To optimize the variables, coordinate descent is used. Coordinate descent is optimizing one latent variable at a time while temporarily keeping the others fixed. This makes the function for $U$ and $V$ become separate square functions with regards to $R$ and can therefore be updated by alternating least squares in the following way:

$$u_i \leftarrow (V_i V_i^T + \lambda_U I_K)^{-1} V_i R_i$$

$$v_j \leftarrow (U_j U_j^T + \lambda_V I_K)^{-1} (U_j R_j + \lambda_V \text{CNN}(W,X_j))$$

Where $u_i$ and $v_j$ are the latent vector that are optimized. In the equation that optimizes $u_i, V_i$ is the set of item latent vectors that user $u_i$ has rated and $R_i$ is the set
ratings given by user $u_i$ to those items. In the second equation that optimizes $v_j$, $U_j$ is the set of user latent vectors that has rated item $v_j$ and $R_j$ is the set of ratings that item $v_j$ has gotten from those users. In the second equation there is also another term $\lambda \nu CNN(W, X_j)$ which integrates the collaborative model with the CNN.