Master Thesis
Business Analytics

Comparison of Deep Learning Product Recommendation Engines in Different Settings

Author:
Robin Opdam

Master Thesis
Business Analytics

Comparison of Deep Learning Product Recommendation Engines in Different Settings

Robin Opdam
2577474

January 2022

Vrije Universiteit
Amsterdam,
Faculty of Science

De Boelelaan 1081a
1081HV Amsterdam

Supervisor:
Prof. Dr. Guszti Eiben
Second Reader:
Prof. Dr. Ger Koole

Global Strategy,
Analytics and
Execution Firm

De Entree 69
1101BH Amsterdam

Supervisors:
Ashish Dang
Marcello Cacciato
Panayiotis Pantelides

Preface

This thesis has been written to fulfil the requirements for the master Business Analytics at the Vrije Universiteit Amsterdam. The objective of the Business Analytics programme is to enable us to recognise and solve in-company problems by applying a combination of methods based on computer science, mathematics and business management. This master is a multidisciplinary programme consisting of three tracks. This thesis belongs to the computational intelligence component of the master programme. Next to combining academic research with real-life problems, the objective of this thesis is to aid in the research of the internship company and enrich my own understanding of the subject.

The internship has taken place at the Data Science department within YGroup. This component of the company provides the data-driven insights needed in business applications relevant for their clients. As recommender systems are more a necessity than ever before, many of their clients face the production and/or implementation of such systems. With the rise of deep learning and this general desire for recommender systems, this thesis is used to provide the company with insights on different algorithms in different settings. Thus, this thesis focuses on the comparison of deep learning recommender systems and a classic approach for differently structured datasets.

I would like to thank my university supervisor, Prof. Dr. Guszti Eiben, for the support and guidance throughout this research. His setup in which he grouped Business Analytics students provided great support and a different view on each other's work. I believe this contributed to the work of each student in this group. In addition, I would like to thank Prof. Dr. Ger Koole for being the second reader.

From YGroup, I would like to thank my external supervisors Ashish Dang, Marcello Cacciato and Panayiotis Pantelides who have supported me during the fulfilment of this work. The frequent meetings and discussions within the internship period have proven very insightful and meaningful. Finally, having access to YGroup's Paperspace cloud computing services enabled me to explore, build and learn more during this internship.

Executive Summary

Problem Definition: In the current decade of information overload, recommender systems have shifted from being a nice-to-have to a necessity in many industries. This shift together with the current boost of deep learning have led to many novel recommender system algorithms, each with their own characteristics. The challenge is to select the right algorithm in the right setting.
Academic/Practical Relevance: We show how two deep learning collaborative filtering based recommender systems compare to each other in terms of recommendation classification and ranking performance. In addition, we compare the aforementioned systems to a matrix factorisation based approach, all for differently structured datasets containing implicit feedback. This provides more insight into the applicability of the two deep learning approaches and highlights the advantages and shortcomings of each algorithm based on the underlying dataset.
Methodology: Using the publicly available MovieLens 1M dataset and two different subsets of an Amazon fashion dataset we observe the behaviour of the different algorithms during training and in the results. The algorithms are optimised using a grid search per dataset. The difference in performance is analysed based on recall@n and NDCG@n metrics, showing significantly different results between each dataset and algorithm.
Results: We show our implementation of the algorithms results in similar behaviour on the MovieLens 1M dataset as observed in the literature, where the deep learning algorithms clearly outperform the matrix factorisation approach. However, on the fashion datasets we observe vastly different behaviour of the deep learning algorithms. The matrix factorisation model exhibits robust performance on each dataset, dominating the deep learning approaches on one out of three datasets. Mixing the structural characteristics of fashion and movie datasets exposes the potential drawbacks and advantages of each method.
Managerial Implications: With this comparison we reveal important differences between the performance of state-of-the-art recommender systems and a classic approach. These insights can be utilised in attaining the optimal algorithmic fit for differently structured practical applications.

Contents

List of Abbreviations

AdaGrad *Adaptive Gradient Algorithm*. 28, 32

Adam *Adaptive Moment Estimation*. 32

AE *Autoencoders*. 7

ALS *Alternating Least Squares*. 13

BPR *Bayesian Personalised Ranking*. 6–13, 15, 27, 29, 33, 34, 39–42, 46–48, 50–53, 55, 56, 58, 60–67, 82–84

BPR-Opt *Bayesian Personalised Ranking Optimisation: a generic optimisation criterion for optimal personalised ranking*. 15, 17

BPTT *Back Propagation Through Time*. 21, 25

CF *Collaborative Filtering*. 2–4, 7–9, 11, 29, 35, 65

CFRNN *Collaborative Filtering with Recurrent Neural Networks*. 3, 8–11, 26–28, 34, 39–41, 44, 48, 51, 53, 57, 58, 60–67, 82–84

CG *Cumulative Gain*. 42

CNN *Convolutional Neural Networks*. 7, 8

ConvNCF *Convolutional Neural Collaborative Filtering*. 8

DCG *Discounted Cumulative Gain*. 42

DeepFM *Deep Factorisation Machines*. 6

DNN *Deep Neural Networks*. 19, 20, 25, 28, 29, 44

EDA *Exploratory Data Analysis*. 75, 76

FM *Factorisation Machines*. 5, 7, 8

GMF *Generalised Matrix Factorisation*. 12, 29–33, 46, 47, 58–61, 63, 65–67

GRU *Gated Recurrent Units*. 9, 25, 66

LSTM *Long Short-Term Memory*. 8, 9, 18, 23–27, 44, 57, 66

MF *Matrix Factorisation*. 3–9, 11–17, 26, 29, 30, 39, 40, 60, 62–65

MLP *Multi-layer Perceptron*. 6, 7, 9, 12, 29, 31–33, 46, 47, 58–63, 65–67

NCF *Neural Network based Collaborative Filtering*. 3, 6, 8, 9, 11, 29–31, 64

NDCG *Normalised Discounted Cumulative Gain*. 42, 48–50, 53, 57, 59–65, 67

NeuMF *Neural Matrix Factorisation*. 9–11, 29, 31–34, 39–41, 46, 48, 52, 53, 58–67, 82–84

RNN *Recurrent Neural Networks*. 8, 9, 18, 20, 21, 23–27, 44

SGD *Stochastic Gradient Descent*. 13, 14, 17, 27, 28, 30, 33, 56

SVD *Singular Value Decomposition*. 5, 11, 13

Y *YGroup*. 1, 3, 4, 9, 34, 64, 84

Chapter 1

Introduction

1.1 About YGroup

YGroup (Y) is a rapidly growing strategy consultancy firm with more than 300 employees and 8 offices in different countries around the world. The company takes pride in transforming strategies from intuitive- and experience-driven to insight- and data-driven. Two of its founders, previously employed at PricewaterhouseCoopers, desire a different approach to strategy consultancy. Therefore, they spend a significant amount of time and energy on creating a data-driven decision environment. Furthermore, strategy implementation is a must for Y, as it creates long-lasting changes within their client companies.

In order to stay ahead of its competition and offer state-of-the-art solutions to their partners and clients, they conduct their own research. This thesis contributes to Y's in-house research on modern recommendation engines and their applications in fashion recommendation.

1.2 Information Overload

In this decade of information overload, everyone has to cope with a tremendous number of available choices. Within the e-commerce sector millions of options are available per product category. The importance of e-commerce continues to grow, with an estimated increase in worldwide e-commerce sales from $2.29 trillion in 2017 to $4.48 trillion by the end of 2021 (eMarketer, 2017). Now that the COVID-19 pandemic has struck the world, this projection is already an understatement of the accelerated growth of e-commerce. The sector is expected to grow 18% in the U.S. this year, compared to a 14.9% increase in 2019 (Samet, 2020). Recommender systems are an effective tool for businesses in overcoming and utilising this information overload. An increasing number of companies are employing recommender systems to capture the opportunities of over-choice within their customers. Tech giant Amazon has been utilising these

systems as an e-commerce strategy for their online retail since 1990 (Smith & Linden, 2017). Even today, Amazon's recommender system is partly responsible for its continuous success and ever-growing customer base. Not only online retailers but also media companies observe the benefits of implementing these systems. The importance of recommender systems became clear in 2006, when Netflix started a competition for improving their current algorithm, with a grand prize of one million dollars (Hallinan & Striphas, 2016). Nowadays 80% of the movies watched on Netflix were recommended by their algorithm (Gomez-Uribe & Hunt, 2016). On the Google-owned video-sharing platform, Youtube, 60% of its video clicks came from home page recommendations (Davidson et al., 2010).

In contrast, the fashion industry seems to fall behind on the recommender systems trend, but for good reason. With fashion, we imply clothing and accessories available at retailers and online e-commerce channels. The lifetime of items within fashion is short compared to movies or music, which results in a short amount of time for data collection on each item, creating an even more sparse domain. With new daily releases, these algorithms have to be able to adapt as quick as fashion changes. Furthermore, customer's individual preference can change rapidly, driven by changes in for example personality, style, and season. However, with an increasing customer demand for personalised online experience, fashion retailers are now trying to incorporate these systems within their e-commerce platforms.

1.3 Recommender Systems

There are two major paradigms of recommender systems which we cover briefly; collaborative and content-based methods. *Collaborative Filtering* (CF) methods base their predictions on past interactions between users and items to produce new recommendations. We can further divide CF methods into memory-based and model-based approaches, where the former does not assume a model and is essentially based on nearest neighbours search. Model-based approaches, on the other hand, assume a model underlying the interactions between users and items and try to discover this model to make new recommendations. CF methods can make accurate predictions based solely on the user-item interaction matrix, which is their main advantage. Their biggest drawback, however, is the cold start problem: new users/items or users/items with little history in the system cannot efficiently utilise the CF algorithms, as they base their recommendations on historical data.

Naturally, the other set of models are content-based models that use additional information about users and/or items, e.g., sex, age and category to make recommendations. The idea is to build a model that can explain the observed user-item interactions based on the available features. These models suffer less from the cold start problem as the additional information is available from the start. They range from simple classification or regression models to much more

complex deep learning variants.

Combining content-based and CF approaches in hybrid methods utilises more of the available data and can alleviate problems like the cold start problem, as discussed in (Burke, 2002).

Deep learning is a subfield within machine learning that has been gaining popularity, it uses multiple layers to progressively extract high-level features from raw data. Over recent years, deep learning has been successfully applied in many fields, such as natural language processing, computer vision, and information retrieval (Deng, 2014). Recently, the application of deep learning has penetrated the field of recommender systems. Its field of research is flourishing as interests are high from both a business and an academic perspective. Several existing deep learning models have been implemented to recommend products or services (Zhang, Yao, Sun, & Tay, 2019), many of which are discussed in Chapter 2.

1.4 Research Questions

YGroup (Y) desires to utilise recent research to build state-of-the-art fashion recommender systems and to have the right approach for their clients. In particular, they are interested in the use of deep learning in fashion product recommender systems. Therefore, we define the research questions as:

RQ1 How do *Collaborative Filtering with Recurrent Neural Networks* and *Neural Network based Collaborative Filtering* compare to each other in terms of recommendation performance on fashion and movie datasets?

RQ2 How do these deep learning models perform compared to a *Matrix Factorisation* benchmark model in terms of recommendation performance on fashion and movie datasets?

To answer these questions we consider the following sub-questions:

SQ1 What are the structural differences between fashion and movie data?

SQ2 How to measure model performance, and which metric is most suitable for our research?

SQ3 How do the structural differences between the datasets affect model performance?

Much of the existing research is evaluated on rich datasets, such as the MovieLens one million ratings data (*MovieLens 1M data*, 2003). As one can imagine, there exist many structural differences between a dataset of ratings per movie and that of ratings per piece of clothing. We take these differences into account by considering both a fashion rating dataset and a movie rating dataset within our research. We focus solely on model-based CF algorithms for two reasons. First,

based on data insights from the company's clients, it becomes clear that often there is only purchase history data available. This implies that we can only observe which item a user bought and when. Secondly, to negate unfairness in comparison, by only using model-based CF algorithms we keep the difference in data utilisation by the models to a minimum.

Since the fashion e-commerce dataset from Y's client is currently unavailable for this research, we use an open-source dataset from Amazon. The 5-core Amazon Clothing Shoes and Jewellery review dataset (Ni, Li, & McAuley, 2019; *Amazon Review data*, 2018), where 5-core implies that all users and items have at least five reviews each. This dataset consists of product ratings and reviews, which does not coincide with the fashion purchase history data Y is currently working with. Therefore, we interpret each rating of a user as a purchase, resulting in a purchase history 5-core Amazon Clothing Shoes and Jewellery dataset. For the movie dataset we take the MovieLens 1M dataset, as this is a widely known and frequently used dataset for recommender systems. As this dataset also consists of ratings we treat each rating in the same manner, thus, ending up with a MovieLens 1M watch history dataset.

To the best of our knowledge, this research is the first to compare two deep learning algorithms and a *Matrix Factorisation* (MF) benchmark for recommendations on a fashion and a movie dataset. Besides, there is no consistent representation in terms of the evaluation criteria used throughout the research that introduces the aforementioned algorithms. Thus, this work also provides metrics never obtained before for the algorithms considered.

After discussing related work in Chapter 2 we go into detail about the selected algorithms in Chapter 3. Next, the experimental setup in Chapter 4, this chapter answers **SQ1** by going into detail about the datasets and **SQ2** explaining why different performance metrics are used. In Chapter 5 we present the experimental results. Afterwards, the analysis and discussion of the results in Chapter 6 covers **SQ3**. Finally, Chapter 7 concludes this research by answering both research questions and describes which extensions can be considered in future research.

Chapter 2

Related Work

The algorithms utilised in this research are selected from a great body of literature on recommender systems. First, we discuss related *Matrix Factorisation* (MF) techniques and some of their shortcomings for implicit feedback data. Next, the various ways to apply deep learning in recommender systems. At the end of this chapter we summarise the selected models and elaborate on our choice.

2.1 Matrix Factorisation

MF is a class of methods, which involve the decomposition of one matrix into the product of two new matrices. *Singular Value Decomposition* (SVD) is the most popular MF algorithm for predicting ratings from historical data. SVD for movie personal rating prediction came into existence during the Netflix challenge in 2006 (Bennett & Lanning, 2007; Bell, Koren, & Volinsky, 2010). Ever since its creation, this model has been extensively researched and widely adopted in practice. Many extensions, such as weighted ratings, time dependency, implicit feedback, and large scale parallel processing have been introduced to improve overall performance (H. Chen, 2017; Koren, Bell, & Volinsky, 2009; Zhou, Wilkinson, Schreiber, & Pan, 2008). Furthermore, time-dependent preference of customers can be captured within SVD models as described in Koren (2009). As already mentioned, fashion rating matrices tend to be extremely sparse. Therefore, SVD is also used in combination with content-based filtering to achieve better results in fashion recommendations (Kang & Yoo, 2007). A more general class of models called *Factorisation Machines* (FM) can be used as a general predictor, working with any real-valued feature vector. These models combine the advantages of factorisation models and Support Vector Machines (Rendle, 2010).

MF techniques rely heavily on the availability of ratings to model users and items latent factors in a lower-dimensional space. However, in many cases rat-

ings are not available and the model has to make use of implicit feedback. In a movie rating dataset, the implicit feedback could be the fact that the user has watched a movie or not, in e-commerce, whether a user has purchased an item or not. To utilise this information Hu, Koren, and Volinsky (2008) proposed Collaborative Filtering for Implicit Feedback Datasets. In addition to the standard MF approach, they use a measure of confidence for each estimation. This confidence can be any additional information that reflects the preference of users. Even though their method resembles preferences better than plain MF, they still have to consider every item for every user, instead of just the observed items (rated items) as before. *Bayesian Personalised Ranking* (BPR) from Implicit Feedback (Rendle, Freudenthaler, Gantner, & Schmidt-Thieme, 2012) uses stochastic gradient descent with bootstrap sampling and a pairwise loss function to tackle this problem. This method is directly optimised for ranking the recommendations and can be used together with MF and k-nearest neighbours approaches.

2.2 Deep Learning

One of the reasons deep learning is revolutionising recommendation system architectures is because of its ability to capture non-linear and non-trivial user-item relationships. Furthermore, it captures the intricate relationships within the data itself, being able to use visual, contextual and textual information (Zhang et al., 2019). Within deep learning, many frameworks have already been used for recommendation systems.

2.2.1 Multilayer Perceptron

In essence, a *Multi-layer Perceptron* (MLP) or feedforward deep neural network, can be described as a mathematical function that maps a set of input values to output values (Goodfellow, Bengio, & Courville, 2016). In more detail; the input values are transformed in several hidden layers on the forward pass through the network. On the backward pass, the weights within the network are changed according to their gradients, calculated with respect to the loss (prediction errors). These backwards and forward passes continue for several epochs until a preset stopping condition is met. This can be seen as the general setup of deep learning used in many different models.

MLP can be used together with MF techniques to replace the inner product with neural network architecture as in, *Neural Network based Collaborative Filtering* (NCF) (X. He et al., 2017) and Neural Network Matrix Factorisation (Dziugaite & Roy, 2015). Google created Wide & Deep Learning for their play store app recommendations (Cheng et al., 2016). This framework uses wide linear models to capture the direct features from historical data, while the deep neural network captures the abstract representation of the data. Similarly to the intuition behind the Wide & Deep model, *Deep Factorisation Machines*

(DeepFM) (Guo, Tang, Ye, Li, & He, 2017) integrate FM with MLP to model both low- and high-order interactions respectively. He et al. extended the DeepFM framework for sparse predictive analysis (X. He & Chua, 2017) and showed improvement over Google's Wide & Deep Learning model.

2.2.2 Autoencoders

A different way of using deep learning is through *Autoencoders* (AE), first introduced for nonlinear principal component analysis in (Kramer, 1991). These models attempt to reconstruct their input data in the output layer in an unsupervised manner. Which leads to useful feature representations in the most-middle layer of the network. I-AutoRec (Sedhain, Menon, Sanner, & Xie, 2015) is an AE which makes use of a similar objective function as CF approaches to predict item ratings. An extension of I-AutoRec is I-CFN (Strub, Gaudel, & Mary, 2016) which is more robust due to using denoising techniques. Besides, it can incorporate side information in a similar fashion as MF methods. Collaborative Deep Learning is another example of the combination of deep learning with MF. This model uses Stacked Denoising Autoencoders as its perception component and Probabilistic Matrix Factorisation as the task-specific component (Wang, Wang, & Yeung, 2015).

2.2.3 Convolutional Neural Networks

Recommendation data often includes unstructured multimedia data, e.g., images or text, *Convolutional Neural Networks* (CNN) can be used to process such data effectively. In contrast to MLPs, CNNs are deep neural networks that use convolution instead of general matrix multiplication in at least one layer (Goodfellow et al., 2016). Visual Bayesian Personalised Ranking, created and extended by He et al. (R. He & McAuley, 2016b, 2016a), uses a CNN to extract visual features and incorporates them into MF. Important in Bayesian Ranking algorithms for recommendation systems is that they often assume user's preferences are correctly reflected in their implicit feedback, i.e., purchase history, mouse activities, search patterns etc. This influences the way these models are evaluated and does not take actual ratings into account. The work of (Yu et al., 2018) uses CNNs for aesthetic-based clothing recommendation, where the aesthetic features of the clothes are taken into account. This framework also incorporates implicit feedback for optimisation and optimises using BPR. A comparative deep learning model uses two CNNs and one MLP to model the user's image preferences, as explained in Lei, Liu, Li, Zha, and Li (2016). The CNNs process the images, one image the user likes, one image the user dislikes, whereas the MLP processes user information. Before the final layer, the abstract user information is joined with the lower-dimensional image features. Finally, the difference between the two lower-dimensional image and user representations is fed to the cross-entropy loss function.

So far the deep learning frameworks have mostly utilised the rating, user and

image data available, another important source of information is the written review of each user. Deep Cooperative Neural Networks as described in Zheng, Noroozi, and Yu (2017) use two parallel neural networks coupled in the last layers. One network deals with user behaviour while the other learns item properties, all from the written review text per user-item combination. Their steps include creating a review matrix (word embedding) per user, using a convolutional layer to produce new features, and max-pooling to extract the most important parts of the review. The output of the max-pooling layer is fed into a fully connected layer which produces the final outputs for each user and item, based on their text reviews. To bring both models together they concatenate user and item vectors and apply a FM to estimate the corresponding rating. A downside of this model, however, is the fact that review texts might not be available during test time. Also using review text but more similar to Collaborative Deep Learning is the Convolutional Matrix Factorisation model which utilises CNNs to capture the contextual information and integrates this in Probabilistic Matrix Factorisation (Kim, Park, Oh, Lee, & Yu, 2016). A more straightforward approach is using CNNs to improve NCF in *Convolutional Neural Collaborative Filtering* (ConvNCF) (X. He et al., 2018). An important difference, however, is that ConvNCF uses an outer product instead of the usual dot product for modelling the user-item interactions. Afterwards, CNNs are used to obtain high-order correlations among embeddings dimensions. The ConvNCF is a model within their proposed Outer Neural Collaborative Filtering framework, it has 6 convolutional layers with embedding size 64 and follows the half-size tower structure. This model also uses the BPR objective for optimisation.

2.2.4 Recurrent Neural Networks

These types of networks, allow output from previous layers to be used in the current layer and share weights among time steps (Goodfellow et al., 2016). This implies the network has a form of memory, which can be useful in processing sequential data, e.g., speech or voice fragments. In fashion recommendation, *Recurrent Neural Networks* (RNN) can be used to model sequential patterns, e.g., user behaviour, fashion trends, and seasonal evolution of items. An example of this approach is *Collaborative Filtering with Recurrent Neural Networks* (CFRNN) (Devooght & Bersini, 2016), in which they view CF as a sequence prediction problem. Here the authors take a similar approach as with language modelling, using RNNs to learn sequences of words (Kombrink, Mikolov, Karafiát, & Burget, 2011). The catalogue of items represents the vocabulary, making each item similar to a word. Then naturally, the sequence of items consumed by a user becomes a sentence and the model's target is to predict the next 'word'. In C.-Y. Wu, Ahmed, Beutel, Smola, and Jing (2017), the authors propose a Recurrent Recommender Network to predict future behavioural trajectories. In addition to standard MF to learn latent user and item attributes, they make use of a *Long Short-Term Memory* (LSTM) (Hochreiter & Schmidhuber, 1997) autoregressive model to capture dynamics. Jing and Smola (2017) tackle both the problem of when a user will return and what to

recommend for an online music service. This model also utilises LSTM units, but this time for recommending the right item at the right time. In Donkers, Loepp, and Ziegler (2017) it is shown that using an RNN with specialised *Gated Recurrent Units* (GRU) (Cho, van Merrienboer, Gülçehre, et al., 2014), allows for seamless integration of user-related information into their model.

While the algorithms mentioned above assume the history of a user is known, this is not always the case. Many websites do not log the user's historical information or are not allowed to, and have to recommend using their current session. Since this is not within the scope of this research but significant in modern online recommender systems, we briefly cover a number of examples. In most of these session-based cases, an item-to-item recommender is used to still make relevant recommendations. However, Hidasi, Karatzoglou, Baltrunas, and Tikk (2015) argue that by modelling the whole session of a user with an RNN-based approach, they can provide more accurate recommendations. S. Wu et al. (2016) use an RNN in which each hidden layer models how the combination of web pages are accessed and their order. They propose a finite history in which the old states collapse into a single history state. Additionally, there are two extensions which elaborate on the inclusion of side information described in Hidasi, Quadrana, Karatzoglou, and Tikk (2016) and Smirnova and Vasile (2017). Instead of only learning from the history or only using the session's information, the Behaviour-Intensive Neural Network for next-item recommendation incorporates the current session information together with the customers long term preferences (Li et al., 2018). This neural network consists of discriminative behaviours learning with LSTM units and a neural item embedding.

2.3 Summary

As mentioned in Chapter 1, *YGroup* is dealing with purchase history data instead of rating data. Therefore, we cannot utilise the fact that user preferences are reflected in their ratings for the Amazon Fashion and MovieLens data. Furthermore, standard MF cannot deal with implicit feedback efficiently, leading us to adopt the BPR framework (Rendle et al., 2012), which optimises for ranking. Thus, we use MF with BPR optimisation as our non-deep learning CF benchmark. In the rest of this work we refer to BPR-MF as BPR.

The first deep learning recommendation algorithm we adopt is *Collaborative Filtering with Recurrent Neural Networks* (CFRNN) as proposed by Devooght and Bersini (2016). This algorithm treats CF as a sequence prediction problem. We decided on CFRNN as it is still a form of CF and except for time, utilises the same data features as the BPR algorithm.

Secondly, *Neural Matrix Factorisation* (NeuMF) as proposed by X. He et al. (2017), this method combines the linearity of MF with the non-linearity of MLP. It is based on their *Neural Network based Collaborative Filtering* (NCF)

framework and is optimised using a point-wise probabilistic approach.

Note that many different approaches exist to evaluate recommender systems and are used throughout the literature. In the selected research we already observe a difference in training, validation and test splits. In addition, the evaluation metrics and the number of items considered during evaluation also differ.

Both BPR and CFRNN are implemented following the methodology as explained in their respective research. For NeuMF we partly utilise the available implementation on *NCF Framework* (2018). We refer the curious reader to Appendix C for more information on the implementation of each algorithm.

Chapter 3

Algorithm Description

This chapter explains the algorithms used, how they are optimised and how they can be used to create recommendations. Before explaining the *Bayesian Personalised Ranking* (BPR) benchmark we provide the formal notation used throughout this chapter. Furthermore, since many CF algorithms are based on *Matrix Factorisation* (MF) we first describe basic *Singular Value Decomposition* (SVD) for recommendations. After clarifying the basics, we go into the benchmark, followed by *Collaborative Filtering with Recurrent Neural Networks* (CFRNN) and finally *Neural Matrix Factorisation* (NeuMF) under the *Neural Network based Collaborative Filtering* (NCF) framework.

3.1 Notation

Table 3.1 lists the general notation used throughout this chapter, followed by specific notations belonging to each algorithm. Note that both BPR and a part of NeuMF are based on MF, therefore, the overlapping notation is omitted for *Neural Matrix Factorisation* (NeuMF).

Table 3.1: General and algorithm specific formal notation

Notation	Explanation
General	
$\mathcal{U}, \mathcal{I}$	user and item set
$\mathcal{M}, \mathcal{N}$	number of users $\lvert\mathcal{U}\rvert$, number of items $\lvert\mathcal{I}\rvert$
S	observed implicit feedback, $S \subseteq U \times \mathcal{I}$
$\mathcal{I}_u^+$	positive item set for user u, $\{i \in I : (u, i) \in S\}$
r	user-item rating matrix
$r_{u,i}$	actual rating user u gives to item i
$\hat{r}_{u,i}$	predicted rating user u gives to item i
$\mathcal{P}, \mathcal{V}, \mathcal{T}$	training set, validation set, test set
Matrix Factorisation & BPR	
γ	dimension of latent factors
p	$\mathcal{M} \times \gamma$ user latent factor matrix
q	$\mathcal{N} \times \gamma$ item latent factor matrix
λ	regularisation parameter
Θ	MF model used with BPR
L	BPR-Opt loss function
α	learning rate
λ_p	L2 regularisation of p
λ_q	L2 regularisation of q
Recurrent Neural Networks	
$\boldsymbol{U}$	input-to-hidden weight matrix
$\boldsymbol{W}$	hidden-to-hidden weight matrix
$\boldsymbol{V}$	hidden-to-output weight matrix
t	time step $t \in T$
$\boldsymbol{h}^{<t>}$	sum of weighted inputs before activation
$\boldsymbol{a}^{<t>}$	activation function at time t
b_h, b_y	input biases, output biases
$\boldsymbol{o}^{<t>}$	output value
$L^{<t>}$	sum of losses of every time step up until t
α	learning rate
$\Theta(t)$	weights and biases at time t
k	user-item interaction sequence cut-off
Neural Collaborative Filtering	
ϕ_{out}	mapping function for the output layer
ϕ_X	X-th neural collaborative filtering layer
L	binary cross-entropy loss
$\odot$	element-wise product of vectors
$\mathbf{h}$	edge weights of output layer in GMF
a_{out}	activation function of output layer in GMF
p^G, q^G	user and item latent factor matrics in GMF
$\mathbf{W}_x$	x-th layer's weight matrix in MLP
$\mathbf{b}_x$	x-th layer's bias vector MLP
a_x	x-th layer's activation function MLP
p^M, q^M	user ant item factor matrices in MLP

3.2 Singular Value Decomposition

This algorithm was developed for recommendation purposes during the Netflix Prize in 2006, where the winning blend of methods included the so called Singular Value Decomposition++ algorithm together with Restricted Boltzmann Machines. It is important to understand and know SVD's limitations, as it forms the foundation of the MF-based *Bayesian Personalised Ranking* (BPR) benchmark.

SVD is a form of MF in which the $\mathcal{M} \times \mathcal{N}$ user-item rating matrix r is being factorised by user and item latent factor matrices, p and q respectively. Here $\mathcal{M}$ represents the number of users and $\mathcal{N}$ denotes the number of items2577474Y. . Each row within p represents a single user's latent factors, similarly, each row in q represents an item's latent factors. As r can be factorised by p and q the original rating matrix can be rewritten as

$$r = q^T p, \tag{3.1}$$

where $q^T p$ is the dot product of q^T and p. This means p and q are matrices with dimensions $\mathcal{M} \times \gamma$ and $\mathcal{N} \times \gamma$ respectively. Here γ denotes the dimension of latent factors. A visual representation of this decomposition is shown in Figure 3.1. In general, the rating matrix r is very sparse, which rules out actual

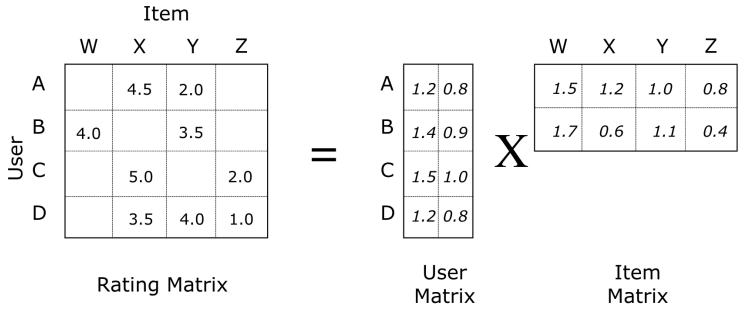


Figure 3.1: Example representation of Matrix Factorisation, where $\mathcal{M}$ and $\mathcal{N}$ equal 4 and γ equals 2 (Liao, 2019)

Singular Value Decomposition as the eigenvectors of r r^T, do not exist (Klema & Laub, 1980). Therefore, we approximate each rating r_{ui} by optimising the latent factor vectors p_u and q_i, resulting in

$$\hat{r}_{\text{ui}} = q_i^T p_u. \tag{3.2}$$

Following the algorithm as described in Koren et al. (2009) we can find p_u and q_i through either *Stochastic Gradient Descent* (SGD) or *Alternating Least Squares* (ALS). For this research we utilise the SGD approach as it combines implementation ease and relatively fast running time. ALS could be beneficial in

cases where, for example, parallelization is an option. With the SGD approach, we compute the associated prediction error e_{ui} using

$$e_{ui} = r_{ui} - q_i^T p_u. \tag{3.3}$$

To optimise p_u and q_i we write this as the following sum of squares minimisation, penalising larger errors more severely

$$\min_{q,p} \sum_{(u,i)\in\mathcal{I}^+} \left(r_{ui} - q_i^T p_u\right)^2, \tag{3.4}$$

where I^+ is the set of user-item pairs for which we know r_{ui}, also known as the positive user-item set. Next, we update the parameters p_u and q_i by a magnitude proportional to a learning rate α in the opposite direction of the gradient (SGD), resulting in

$$\begin{aligned} q_i &\leftarrow q_i + \alpha \cdot (e_{ui} \cdot p_u) \\ p_u &\leftarrow p_u + \alpha \cdot (e_{ui} \cdot q_i)\,. \end{aligned} \tag{3.5}$$

After several of these updates in which we move towards a minimum, we obtain matrices p and q^T of which the dot product approximates the actual ratings in rating matrix r. Meaning the the dot product of p and q^T can be used to fill the missing ratings within r. Naturally, this method heavily overfits on the matrix used for training. Hence, we define basic MF with regularisation as:

$$\min_{q,p} \sum_{(u,i)\in\mathcal{I}^+} (r_{ui} - q_i^T p_u)^2 + \lambda(\|\mathbf{q}_i\|^2 + \|\mathbf{p}_u\|^2), \tag{3.6}$$

where λ is the regularisation parameter. This method of regularisation is also referred to as L2 regularisation as it uses the L2 norm of the vectors to penalise the parameters. With this regularisation, the SGD updates are defined as

$$\begin{aligned} p_u &\leftarrow p_u + \alpha\,(e_{ui} \cdot q_i - \lambda_p p_u) \\ q_i &\leftarrow q_i + \alpha\,(e_{ui} \cdot p_u - \lambda_q q_i). \end{aligned} \tag{3.7}$$

With this mechanism in place the magnitudes of q_i and p_u are penalised resulting in a more general model. As mentioned in Chapter 2, many extensions exist which can increase this algorithm's prediction accuracy. With this setup to approximate the actual rating matrix r, we can recommend products to user u for which $q^T p_u$ results in a high rating.

3.3 Bayesian Personalised Ranking

Since we are dealing with purchase history data (binary) instead of ratings (ordinal) we require a different modelling approach. The fact that a user bought a product or watched a movie does not guarantee a preference for that item. In

addition, items the user has not interacted with do not necessarily imply a disliking towards that item. Thus the preference of a user towards non-interacted (negative) items is unknown. Even if we do assume that a purchased item is a preferred item, the standard MF now has to consider every item, the observed and the non-observed. Standard MF becomes infeasible, considering the average rating matrix as used in the literature, consists of millions of possible user-item combinations. Examples of such datasets include the *MovieLens 1M data* (2003) and the *Netflix 100M data* (2019).

Rendle et al. (2012) propose *Bayesian Personalised Ranking* (BPR) with *Bayesian Personalised Ranking Optimisation: a generic optimisation criterion for optimal personalised ranking* (BPR-Opt). This framework delegates the actual modelling of the user-item relationship to an MF or adaptive k-nearest neighbours model. The way BPR-Opt differs from standard MF optimisation is that instead of minimising the differences between predicted ratings and actual ratings, it considers the ranking of item pairs per user. The goal is to find each user's total ranking $>_u \subset \mathcal{I}^2$, where $>_u$ has to meet the following properties of total order:

$$\begin{aligned} &\forall i,j \in I : i \neq j \Rightarrow i >_u j \vee j >_u i && (totality) \\ &\forall i,j \in I : i >_u j \wedge j >_u i \Rightarrow i = j && (antisymmetry) \\ &\forall i,j,k \in I : i >_u j \wedge j >_u k \Rightarrow i >_u k && (transitivity) \end{aligned} \tag{3.8}$$

Their model is based on the assumption that the user prefers a positive item (observed) over a negative item (non-observed), resulting in the implicit feedback representation shown in Figure 3.2. For training we draw user-specific triples from the data, consisting of user u, positive item i and negative item j. Formally, we create the triples $D_S : \mathcal{U} \times \mathcal{I} \times \mathcal{I}$ as

$$D_S := \left\{ (u,i,j) | i \in \mathcal{I}_u^+ \wedge j \in \mathcal{I} \backslash \mathcal{I}_u^+ \right\}, \tag{3.9}$$

where S is the set of observed implicit feedback, $\mathcal{I}$ is the set of all items and $\mathcal{I}_u^+$ is the set of positive items of user u (see Table 3.1 for notation).

3.3.1 BPR-Opt & BPR Learning

Now using a Bayesian analysis of the problem we can formulate the likelihood function as $p\left(i >_u j | \Theta\right)$ with prior $p(\Theta)$, where Θ is the utilised model. For this research, we take standard MF as Θ, meaning MF is used to capture the relationships between users and items. The goal is to maximise the posterior probability, defined as

$$p\left(\Theta | >_u\right) \propto p\left(>_u | \Theta\right) p(\Theta), \tag{3.10}$$

where $>_u$ is the desired latent preference structure for user u. To obtain a general formulation for all users $u \in \mathcal{U}$ we assume users act independently of each other. Furthermore, the ordering of each user-specific item pair (i,j)

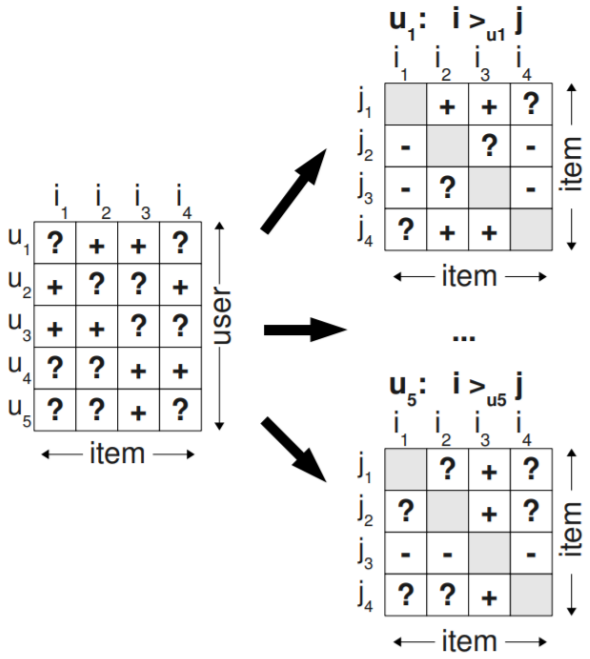


Figure 3.2: BPR assumption on implicit feedback data per user. In the matrices on the right side, (+) indicates users preference of item i over j and (−) indicates users preference of j over i. (Rendle et al., 2012)

is independent of the ordering of every other pair. Using these assumptions together with the totality and antisymmetry property (Equation 3.8) Rendle et al. (2012) define the following Bayesian formulation

$$\prod_{u \in U} p\left(>_u |\Theta\right) = \prod_{(u,i,j) \in D_S} p\left(i >_u j|\Theta\right). \tag{3.11}$$

The individual probability that user u prefers item i over item j can now be defined as

$$p\left(i >_u j|\Theta\right) := \sigma\left(\hat{x}_{uij}(\Theta)\right), \tag{3.12}$$

where $\hat{x}_{uij}(\Theta)$ is defined as the difference between $\hat{x}_{ui}$ and $\hat{x}_{uj}$, calculated with Θ being standard MF and σ denoting the logistic sigmoid function:

$$\sigma(x) := \frac{1}{1 + e^{-x}}. \tag{3.13}$$

Finally BPR-Opt is derived as:

$$\begin{aligned} \text{BPR} - \text{OPT} &:= \ln p\left(\Theta| >_u\right) \\ &= \ln p\left(>_u |\Theta\right) p(\Theta) \\ &= \ln \prod_{(u,i,j)\in D_S} \sigma\left(\hat{x}_{uij}\right) p(\Theta) \\ &= \sum_{(u,i,j)\in D_S} \ln \sigma\left(\hat{x}_{uij}\right) + \ln p(\Theta) \\ &= \sum_{(u,i,j)\in D_S} \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta \|\Theta\|^2. \end{aligned} \tag{3.14}$$

For BPR learning, we use SGD similar to standard MF however, each update is now calculated as follows:

$$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \Theta \right). \tag{3.15}$$

To apply this update using MF, we have to take into account that $\hat{x}_{uij}$ is defined as $\hat{x}_{ui} - \hat{x}_{uj}$. Using the MF formula as shown in Section 3.2 we obtain

$$\hat{x}_{uij} = q_i^T p_u - q_j^T p_u. \tag{3.16}$$

Meaning the partial derivatives needed for the SGD updates of Θ take the following forms:

$$\frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} (q_i - q_j) & \text{if } \theta = p_u, \\ p_u & \text{if } \theta = q_i, \\ -p_u & \text{if } \theta = q_j, \\ 0 & \text{else} \end{cases} \tag{3.17}$$

For the regularisation parameters used within the updates, we define λ_p for user features p and λ_q for item features q.

Optimiser: Bold Driver

In addition to Rendle et al. (2012), we optimise the learning rate during training for better performance using the bold-driver approach (Shepherd, 2012). This simple yet effective method can be described as

$$\alpha_{k+1} = \begin{cases} \rho \alpha_k, & \text{if } L_{k+1} < L_k, \\ \sigma \alpha_k, & \text{if } L_{k+1} \geq L_k, \end{cases} \tag{3.18}$$

where α_k, ρ and σ are the learning rate at iteration k, rate of increase and rate of decrease respectively. Furthermore, L_k is the BPR-Opt loss as defined in 3.14, at iteration k. This method increases the learning rate with $\rho > 1$ at the end of an iteration if the current loss is smaller than the loss in the previous iteration. If the current loss is smaller or equal to the previous loss we decrease the learning rate with $\sigma < 1$.

Recommending

Recommendations for user u are made by taking the dot-product of trained vector p_u and matrix q, which results in predicted preference scores for all items. Within these scores a larger score signifies more preference towards that item. The top n items with the largest scores are the items we recommend to user u.

3.4 Collaborative Filtering with Recurrent Neural Networks

Recurrent Neural Networks (RNN) are already well known when it comes to text generation, where the next word is predicted given the current sequence of words (Kombrink et al., 2011; Sutskever, Martens, & Hinton, 2011). In a similar fashion, we can treat product recommendation as a sequence prediction problem. Now the past user-item interactions are modelled as chronologically ordered sequences per user, i.e., their interaction history. Our recommendation is then defined as the predicted interaction, given the user's item sequence. We adopt the approach of Devooght and Bersini (2016) in which they propose implementing an RNN with one hidden *Long Short-Term Memory* (LSTM) layer to model the recommendation problem as a sequence prediction problem. Before elaborating on their approach we go into the basics of feedforward neural networks followed by an in-depth description of RNNs and LSTMs, explaining why these models are a good fit for sequence prediction.

3.4.1 Feedforward Neural Networks

In general, feedforward neural networks consist of an input layer, a number of hidden layers and the output layer. Each layer consists of several neurons that are connected with weights. The neurons in the hidden layers use activation functions to transform the combined input and weights from the previous layer in a non-linear way. This non-linear activation can be seen as a transformation of a simple linear regression. As an example we take input vector X, bias b, weight vector W and estimation of desired output vector y, $\hat{y}$, we create the simple linear regression equation

$$\hat{y} = b + W^\top X. \tag{3.19}$$

Here we optimise for the best fit of the line through the data by minimising a loss function, such as the mean squared error. Now without optimal weights and bias terms, there will be a significant difference between our estimation $\hat{y}$ and the real value(s) y. In other words, sub-optimal parameters lead to a larger loss than optimal parameters. In a single-layered single neuron feedforward neural network we have activation function a in place

$$\hat{y} = a(b + W^\top X). \tag{3.20}$$

This model is also known as a perceptron model as introduced in Rosenblatt (1957), visually represented in Figure 3.3. Compared to linear regression, the

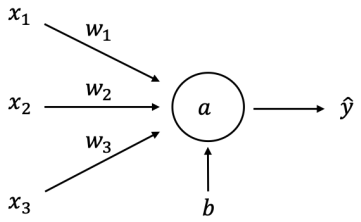


Figure 3.3: A single neuron, single-layered neural network, also known as a perceptron model

parameters of this non-linear variant do not act independently of each other when it comes to influencing the loss function. By introducing this non-linearity and allowing the model more flexibility, we trade the convex solution space of linear regression for one with many local optima. For the model to still perform well and be able to find a satisfying optimum we need multi-step optimisation methods. The most effective method to date is known as gradient descent, which is used in a similar fashion as in Section 3.3. However, with the non-trivial solution space at hand, *Deep Neural Networks* (DNN) usually perform mini-batch stochastic gradient descent. This method splits the full training batch in smaller mini-batches which enlarge the update frequency compared to batch (all training samples) gradient descent. The stochastic component is introduced by updating the parameters per mini-batch instead of after the full batch has gone through the network.

Taking the single-layered single neuron example, the input with its randomly initialised weights and bias go through Equation 3.20 to calculate estimation $\hat{y}$. This forward step, where we move from left to right through Figure 3.3 is known as forward propagation. Next, we update the parameters (weights and bias) by a magnitude proportional to the learning rate α in the opposite direction of the partial derivatives of the loss function (gradient descent). Thus we take a step backwards through the network, towards the inputs, which is known as backward propagation. One forward and backward pass are defined as an epoch, these propagation steps continue for several epochs or until a stopping condition is met. Expanding this example to DNNs, we have several hidden layers, consisting of multiple neurons, each with their own input and output connections.

Sequence Prediction

A general representation of a feedforward neural network is shown in figure 3.4, here the internal connections are omitted and the inputs and outputs are represented as sequences (as in our problem). In our case input data X consists

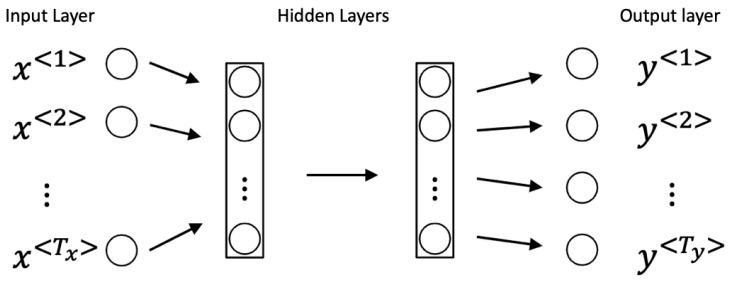


Figure 3.4: General representation of a Neural Network with a sequence as input data and a sequence as output data (Bhulai, 2018)

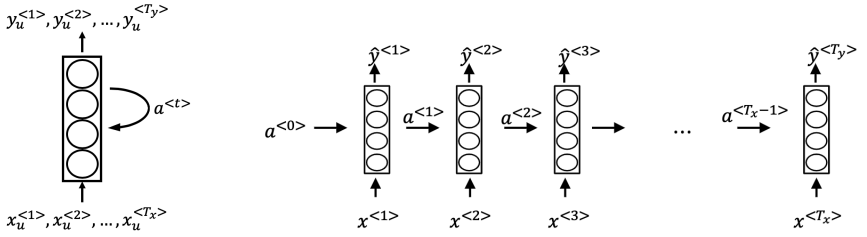
of sequences, meaning every $x_u \in X$ is a sequence $x_u = x_u^{<1>}, x_u^{<2>}, ..., x_u^{<T>}$, with time steps $t = 1, 2, ..., T$. Modelling the recommendation problem as a sequence prediction problem means we assume the items in each sequence are not independent observations. For each user, the previously bought items contain information about the next item, similar to words in a sentence for text generation problems. If we model this using standard feedforward DNNs where each $x_u^{<t>}$ is one input node, the model would have separate parameters for each input node (Figure 3.4). This implies that the network needs to learn all underlying rules of the sequence separately, for every position in the sequence (Goodfellow et al., 2016). Even if this network would learn to identify the important parts of the sequence, it will be tailored towards the input sequence and unable to generalise.

Thus, we need the network to remember inputs from previous time steps instead of processing the full sequence all at once. This is achieved in RNNs by sharing parameters across the time steps of the input sequence.

3.4.2 Recurrent Neural Networks

With RNNs, the input is fed into the network one time step at a time, while retaining information from previous inputs. Therefore, we can produce a prediction $\hat{\mathbf{y}}^{<\mathbf{t}>}$ at every time step given the current input $\mathbf{x}^{<\mathbf{t}>}$ and the previous inputs as shown in Figure 3.5.
Note that every arrow in Figure 3.5b represents a weight matrix, one for each input and one for the output per time step. Before going into the formal notation of the forward and backward propagation we specify $\boldsymbol{U}$, $\boldsymbol{W}$ and $\boldsymbol{V}$ as the weight matrices for input-to-hidden, hidden-to-hidden and hidden-to-output layers respectively.

In the case of a vanilla RNN unit as shown in Figure 3.6, the hyperbolic tangent activation function is used (also known as *tanh*) (Nwankpa, Ijomah, Gachagan, & Marshall, 2018). This differentiable function maps the input value to a value



(a) Folded (b) Unfolded

Figure 3.5: General Representation of a Recurrent Neural Network (Bhulai, 2018)

between minus one and one. With the *tanh* and the previously defined weight matrices $\boldsymbol{U}$, $\boldsymbol{W}$ and $\boldsymbol{V}$, we define forward propagation at time step t as:

$$\boldsymbol{h}^{<t>} = b_h + \boldsymbol{W}\boldsymbol{a}^{<t-1>} + \boldsymbol{U}\boldsymbol{x}^{<t>} \quad [3.21]$$

$$\boldsymbol{a}^{<t>} = \tanh(\boldsymbol{h}^{<t>}) \quad [3.22]$$

$$\boldsymbol{o}^{<t>} = b_y + \boldsymbol{V}\boldsymbol{a}^{<t>} \quad [3.23]$$

$$\hat{\boldsymbol{y}}^{<t>} = \text{softmax}(\boldsymbol{o}^{<t>}) \quad [3.24]$$

Where $\boldsymbol{h}^{<t>}$ is the sum of the weighted inputs before activation in $\boldsymbol{a}^{<t>}$. Furthermore, the input and output biases are represented by b_h and b_y. To obtain the output per time step we apply a softmax activation which maps the output value $\boldsymbol{o}^{<t>}$ to a value between zero and one. This value can be interpreted as a probability, allowing us to calculate a loss value $L^{<t>}$ based on the negative log-likelihood of $\hat{y}^{<t>}$ given $x^{<1>}, x^{<2>}, ..., x^{<t>}$. Since every input produces an output value here (Figure 3.5), the total loss is defined as the sum of losses of every time step:

$$\begin{aligned} \sum_t L^{(t)} &= L(\{x^{<1>}, ..., x^{<t>}\}, \{y^{<1>}, ..., y^{<t>}\}) \\ &= -\sum_t \log p_{\text{model}} \left(y^{<t>} | \{x^{<1>}, \ldots, x^{<t>}\}\right) \end{aligned} \quad [3.25]$$

Here p_{model} is the loss of output value $\hat{y}^{<t>}$ on the actual observation $y^{<t>}$, given the inputs up until time t.

Equations 3.21-3.25 account for the forward pass of an RNN with vanilla hidden units. Now for the back propagation we have to take time into account, resulting in *Back Propagation Through Time* (BPTT). Starting at the final time step we move backwards to the initial time step, while calculating the gradients and updating the parameters at every step. Calculating these gradients means taking the derivatives of the time dependent parameters $\boldsymbol{x}^{<t>}, \boldsymbol{a}^{<t>}, \boldsymbol{o}^{<t>}$, $L^{<t>}$ and the shared parameters $b_h, b_y, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}$, with respect to the loss L. Using Goodfellow et al. (2016) we show the formulas of the partial derivatives, starting

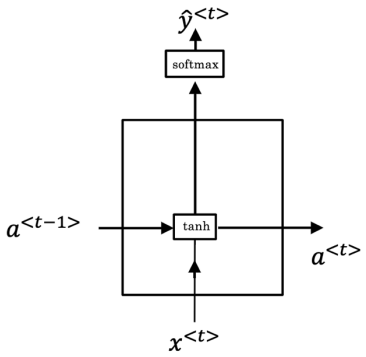


Figure 3.6: Vanilla RNN unit (Bhulai, 2018)

with $\frac{\partial L}{\partial o_i^{<t>}}$

$$(\nabla_{o^{<t>}} L)_i = \frac{\partial L}{\partial o_i^{<t>}} = \frac{\partial L}{\partial L^{<t>}} \frac{\partial L^{<t>}}{\partial o_i^{<t>}} = \hat{y}_i^{<t>} - 1_{i=y^{<t>}}, \tag{3.26}$$

where $\frac{\partial L}{\partial o_i^{<t>}}$ is formulated given the softmax activation is used for obtaining $\hat{\boldsymbol{y}}$. Furthermore we assume the negative log-likelihood is used to calculate the loss on the true targets y. In the first backward time step $\boldsymbol{a}^{<\boldsymbol{T}>}$ has only the final output $\boldsymbol{o}^{<\boldsymbol{T}>}$ as its descendent, making the partial derivative relatively simple:

$$\nabla_{\boldsymbol{a}^{<T>}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{<T>}} L \tag{3.27}$$

For the other time steps we have to take both $\boldsymbol{o}^{<\boldsymbol{t}>}$ and $\boldsymbol{a}^{<\boldsymbol{t+1}>}$ into account:

$$\begin{aligned} \nabla_{\boldsymbol{a}^{<t>}} L &= \left(\frac{\partial \boldsymbol{a}^{<t+1>}}{\partial \boldsymbol{a}^{<t>}}\right)^\top (\nabla_{\boldsymbol{a}^{<t+1>}} L) + \left(\frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{a}^{<t>}}\right)^\top (\nabla_{\boldsymbol{o}^{<t>}} L) \\ &= \boldsymbol{W}^\top \operatorname{diag}\left(1 - \left(\boldsymbol{a}^{<t+1>}\right)^2\right) (\nabla_{\boldsymbol{a}^{<t+1>}} L) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{<t>}} L) \end{aligned} \tag{3.28}$$

Here $\operatorname{diag}\left(1 - \left(\boldsymbol{a}^{<t+1>}\right)^2\right)$ stands for the Jacobian of the *tanh* function associated with the hidden unit i at time $t+1$.

Now for the shared parameters we refer to copies $\boldsymbol{W}^{<\boldsymbol{t}>}$ of $\boldsymbol{W}$, instead of just $\boldsymbol{W}$. This is due to specifics of the $\nabla_{\boldsymbol{W}} f$, which are omitted here (see subsection 6.5.4 of Goodfellow et al. (2016)). The important part is that $\nabla_{\boldsymbol{W}^{<t>}}$ is used to denote the contribution of the weights at time step t to the gradient. Using

this notation for all shared parameters, we obtain the following formulations:

$$\nabla_{b_y} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{<t>}}{\partial b_y} \right)^\top \nabla_{\boldsymbol{o}^{<t>}} L = \sum_t \nabla_{\boldsymbol{o}^{<t>}} L \tag{3.29}$$

$$\nabla_{b_h} L = \sum_t \left(\frac{\partial \boldsymbol{a}^{<t>}}{\partial b_h^{<t>}} \right)^\top \nabla_{\boldsymbol{a}^{<t>}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{a}^{<t>}\right)^2\right) \nabla_{\boldsymbol{a}^{<t>}} L \tag{3.30}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{<t>}} \right) \nabla_{\boldsymbol{V}^{<t>}} o_i^{<t>} = \sum_t \left(\nabla_{\boldsymbol{o}^{<t>}} L\right) \boldsymbol{a}^{<t>\top} \tag{3.31}$$

$$\begin{aligned} \nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial a_i^{<t>}} \right) \nabla_{\boldsymbol{W}^{<t>}} a_i^{<t>} \\ &= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{a}^{<t>}\right)^2\right) \left(\nabla_{\boldsymbol{a}^{<t>}} L\right) \boldsymbol{a}^{<t-1>\top} \end{aligned} \tag{3.32}$$

$$\begin{aligned} \nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial a_i^{<t>}} \right) \nabla_{U^{<t>}} a_i^{<t>} \\ &= \sum_t \operatorname{diag}\left(1 - \left(a^{<t>}\right)^2\right) \left(\nabla_{a^{<t>}} L\right) x^{<t>\top} \end{aligned} \tag{3.33}$$

This concludes the forward and backward propagation of an RNN with vanilla units.

Vanishing or Exploding Gradients

Even though RNNs possess memory in terms of their shared parameters, the use of this vanilla unit introduces an issue when it comes to long term dependencies. We explain this problem using a simplified, linear form of Equation 3.21 without inputs $\boldsymbol{x}^{<t>}$:

$$\boldsymbol{a}^{<t>)} = \left(\boldsymbol{W}^{<t>}\right)^\top \boldsymbol{a}^{<0>}. \tag{3.34}$$

As one can imagine $\boldsymbol{W}^{<t>}$ is unstable, meaning the term either vanishes or explodes depending on the magnitude of $\boldsymbol{W}$ and the time step t. RNNs with vanilla units build on this same principle, meaning we observe similar behaviour of their gradients where they can either vanish or explode (Hochreiter, 1998). As also discussed in Hochreiter (1998), replacing the vanilla RNN units with *Long Short-Term Memory* (LSTM) can mitigate the vanishing/exploding gradients issue.

3.4.3 Long Short Term Memory Units

Within the RNNs hidden layers, there exist different units to use for the modelling of each neuron. LSTM units were introduced as an alternative to the vanilla RNN neurons, as the latter suffered from vanishing/exploding gradients. As the name suggests, these *Long Short-Term Memory* (LSTM) units try to preserve previously observed information while forgetting unnecessary information. This

subsection explains the mechanisms behind LSTM units, following Goodfellow et al. (2016) and Olah (2015).

Next to the recurrence of RNNs, LSTM units introduce a loop within themselves. The information within this loop is regulated with gates, or "*the weight on this self-loop is conditioned on the context, rather than fixed*" (Goodfellow et al., 2016, p. 404). The architecture of a single LSTM unit is shown in Figure 3.7, where each gate is based on the sigmoid function. Furthermore, every gate can be thought of as a layer of its own, combining its own weights and bias with the output. We observe similar inputs and outputs compared to the vanilla RNN

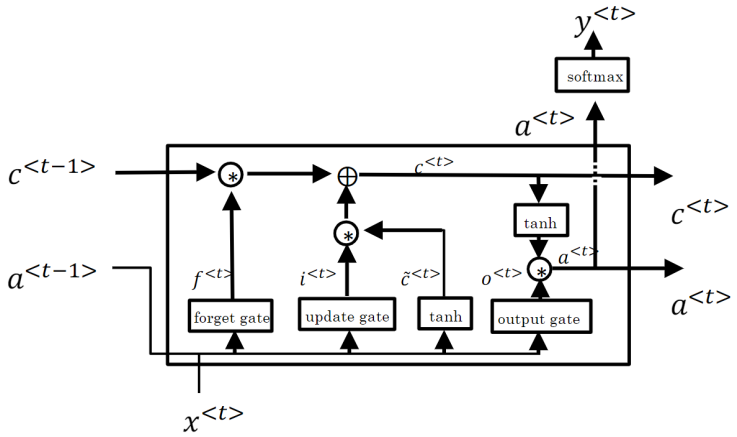


Figure 3.7: LSTM unit architecture (Bhulai, 2018)

unit in Figure 3.6. However, the key difference is the cell state $c^{<t>}$, which runs horizontally through the complete network, connecting the individual LSTM steps throughout time. This cell state can be seen as the main information flow throughout the network of which the gates decide what to keep. As with the vanilla units, we receive the activated output of the previous time step $a^{<t-1>}$ as input. Before going through the gates, $a^{<t-1>}$ is combined with the current input variable(s) $x^{<t>}$.

The **Forget Gate** (3.35) is the first gate we encounter, it consists of a sigmoid activation function, meaning its outputs will be between zero and one. This output $f^{<t>}$ is multiplied with the cell state; therefore, values closer to zero will also lower their corresponding value in the cell state. Naturally, values closer to one mean they have more importance and will be better preserved within the cell state.

$$f^{<t>} = \sigma\left(W_f \cdot \left[a^{<t-1>}, x^{<t>}\right] + b_f\right) \tag{3.35}$$

Next, the **Update Gate** (3.36), which decides what new information to store in the cell state. The sigmoid output $i^{<t>}$ is combined with a *tanh* transformation

of the input $\tilde{C}^{<t>}$. Within this combination $\tilde{C}^{<t>}$ contains the new candidate values that could be added to the cell state. The final decision as to how much each value will be updated is done by scaling $\tilde{C}^{<t>}$ with $i^{<t>}$. After multiplying and adding the forget and update gates respectively, we obtain the final cell state $C^{<t>}$ as is defined in Equation 3.37.

$$\begin{aligned} i^{<t>} &= \sigma\left(W_i \cdot \left[a^{<t-1>}, x^{<t>}\right] + b_i\right) \\ \tilde{C}^{<t>} &= \tanh\left(W_C \cdot \left[a^{<t-1>}, x^{<t>}\right] + b_C\right) \end{aligned} \tag{3.36}$$

$$C^{<t>} = f^{<t><t-1>} + i^{<t>} \cdot \tilde{C}^{<t>} \tag{3.37}$$

Finally, the **Output Gate** (3.38) combines the updated cell state with the current inputs. We first apply a sigmoid function to the inputs, denoted as $o^{<t>}$, to decide what values to keep. Secondly, we squeeze the values of the updated cell state between minus one and one with the *tanh* function. The final output $a^{<t>}$ is the multiplication of the two, similar to the update gate mechanism.

$$\begin{aligned} o^{<t>} &= \sigma\left(W_o\left[a^{<t-1>}, x^{<t>}\right] + b_o\right) \\ a^{<t>} &= o^{<t>} \cdot \tanh\left(C^{<t>}\right) \end{aligned} \tag{3.38}$$

Since each component is constructed using the well known differentiable sigmoid and *tanh* functions, we can still upgrade the weights during BPTT. The additive structure of the gradients concerning the cell state and the presence of the forget gate make it less likely that gradients will vanish during BPTT. This additive structure of the derivative of the cell state looks as follows:

$$\begin{aligned} \frac{\partial C^{<t>}}{\partial C^{<t-1>}} &= \frac{\partial}{\partial C^{<t-1>}}\left[C^{<t-1>} \cdot f^{<t>} + \tilde{C}^{<t>} \cdot i^{<t>}\right] \\ &= \frac{\partial}{\partial C_{t-1}}\left[C^{<t-1>} \cdot f^{<t>}\right] + \frac{\partial}{\partial C^{<t-1>}}\left[\tilde{C}^{<t>} \cdot i^{<t>}\right] \\ &= \frac{\partial f^{<t>}}{\partial C_{t-1}} \cdot C_{t-1} + \frac{\partial C^{<t-1>}}{\partial C^{<t-1>}} \cdot f^{<t>} + \frac{\partial i^{<t>}}{\partial C^{<t-1>}} \cdot \tilde{C}^{<t>} + \frac{\partial \tilde{C}^{<t>}}{\partial C^{<t-1>}} \cdot i^{<t>} \end{aligned} \tag{3.39}$$

As the backward steps through the network are derived in the same manner as with vanilla units, the backpropagation formulas are omitted. For the full derivation we refer the curious reader to G. Chen (2016).

Another popular alternative for vanilla RNN units are *Gated Recurrent Units* (GRU) (Cho, van Merrienboer, Bahdanau, & Bengio, 2014). Since no GRUs have been utilised in this research we keep their explanation out of scope. However, we refer the curious reader to a comparison of the vanilla units, GRUs and LSTMs in Chung, Gülçehre, Cho, and Bengio (2014).

Overview

Concluding this in-depth explanation of RNNs and LSTMs units, we note that the memory introduced by sharing of parameters allowed DNNs to be used in

sequence modelling. However, in its early stages, this shared memory principle through time faced several drawbacks including the vanishing and exploding gradients problem. To mitigate these issues Hochreiter (1998) introduced the LSTM unit, which is a more complex structured unit than the vanilla RNN unit. LSTM units contain a cell-state which can be seen as the main information flow of the network. The information within this cell-state is regulated by gates, based on sigmoid and *tanh* functions.

3.4.4 RNN for Collaborative Filtering

Devooght and Bersini (2016) frame the recommendation problem as a sequence prediction task in which the order of previous interactions is taken into account when predicting the next interaction. With this approach comes the distinction between short-term and long-term predictions, where the former means the very next item in the sequence. The order of user-item interactions can contain valuable information that is neglected in common modelling of the problem, i.e., standard MF. Their model is named *Collaborative Filtering with Recurrent Neural Networks* (CFRNN).

Adopting this approach, we use an RNN to go through each time step of the sequence of items consumed by a user. The input per time step consists of the one-hot encoding of the current item, out of all items. As for the output we use a softmax layer with a neuron for each item, meaning we treat this as a multiclass-classification output. The loss is then calculated according to Equation 3.40, also known as categorical cross-entropy loss or softmax loss.

$$L(y, \hat{y}) = -\frac{1}{\mathcal{M}} \sum_{u=0}^{\mathcal{M}} \sum_{i=0}^{\mathcal{N}} \left(y_{ui} * \log\left(\hat{y}_{ui}\right)\right) \qquad [3.40]$$

Where $\hat{y}$ is the predicted value, $\mathcal{M}$ is the number of users and $\mathcal{N}$ is the total number of items. As every time step produces an output, namely the predicted next item in the sequence, we calculate this loss at each time step. The categorical cross-entropy loss function compares the softmax activated output (probabilities) with the true one-hot encoded distribution. Put differently, the closer the predicted probabilities per class ($\hat{y}_{ui}$) are to the actual single next value (y_{ui}) the lower the loss. This implies that the model is trained to predict the very next item in the sequence, thus focusing on short-term rather than long-term predictions. The loss of a single epoch is computed by taking the average of Equation 3.40 for all time steps of all users. Furthermore, we utilise LSTM units as the hidden neurons based on the difference in performance with vanilla RNN units as explained in Section 3.4.

Important in this approach is that not all user-item interaction sequences are of equal length. To still enable the LSTM units to learn from these sequences, we use the k latest interactions together with padding and masking of the interactions for users with less than k interactions. Thus, the time ordering per user

is relative, meaning users within each batch can have a variable time horizon over which the interactions took place. Furthermore, the number of user-item interactions is not equal for all users, meaning one batch can contain users with different total time steps T.

Diversity Bias

When using multiclass-classification output we expose the model more to the imbalanced implicit feedback dataset compared to BPR. The difference is that the RNN does not build a representation per user and item to update at every interaction encountered, as in BPR. It does, however, use the sequence information per user to predict the next item. This implies that the frequent occurrence of popular items in user-item sequences together with the model optimised to predict the next item leads to the development of a bias towards these popular items during training. In order to mitigate the effects of this popularity bias, we utilise a diversity bias within the objective function, following Devooght and Bersini (2016) we get

$$L_\delta = -\frac{\log\left(o_{\text{correct}}\right)}{e^{\delta p_{\text{correct}}}}, \tag{3.41}$$

where $\delta \in [0, inf)$ is the diversity bias hyperparameter, $o_{correct}$ is the value of the output neuron corresponding to the correct item and $p_{correct}$ denotes a popularity measure associated with the correct item. To construct p we divide the items into ten bins of logarithmic size in which the smaller bins contain the most popular items in terms of the number of ratings. Naturally, the larger bins contain the least popular items. Now we assign a p of 1 to the items in the largest bin, $p = 2$ for items in the second-largest bin, up to $p = 10$ for the smallest bin. Setting $\delta > 0$ ensures the loss for mispredicting the most popular items weighs less than mispredicting less popular items. This way the SGD updates of the parameters with respect to the loss will be less focused on getting the popular items correct and reduces the popularity bias.

Model Structure

Note that before the input is masked and fed into the LSTM layer, we use an embedding layer to densely represent the items instead of using a sparse representation. In other words, we obtain a similar matrix as q in 3.3 in terms of an abstract representation of the item that is learned during training. However, the embedding layer in CFRNN learns the position of an item within the vector space from the sequences and the surrounding items observed during training, which is different from the way BPR utilises the item latent factor matrix q. Finally, the model has the following structure:

1. Embedding layer
2. Masking layer
3. LSTM layer

4. Dense layer (softmax)
5. Categorical Crossentropy Loss with Diversity Bias

Optimisation: AdaGrad

The standard SGD updates for the weights and biases θ at each epoch τ (or each mini-batch) can be described as

$$\theta(\tau+1) = \theta(\tau) - \alpha\frac{\partial L}{\partial \theta}(\tau), \tag{3.42}$$

where α denotes the learning rate and L denotes the loss. Now we define the difference in the weights per epoch number τ as

$$\Delta\theta = -\alpha\frac{\partial L}{\partial \theta}(\tau),\ \textit{where}\ \Delta\theta = \theta(\tau+1) - \theta(\tau). \tag{3.43}$$

One shortcoming of this approach is the fact that all parameters are updated according to the same learning rate at each step. As seen in the Bold-Driver approach in Section 3.3.1 we decrease the learning rate when needed to slow down learning and not overshoot the minimum. However, in DNN updates we observe different frequencies by which each weight is updated while training, especially when the gradients are sparse. If we decrease the learning rate at an equal pace for each weight we might miss the optimal setting per weight. Therefore, we utilise the *Adaptive Gradient Algorithm* (AdaGrad) (Duchi, Hazan, & Singer, 2011) as the learning optimisation algorithm. This approach assigns individual learning rates to the parameters at each step $\theta_i(\tau)$ and adapts these rates during training based on each parameter's update frequency. Or more formally, the difference in weights is calculated using

$$\begin{aligned} \Delta\theta_i(\tau) &= -\frac{\alpha}{\sqrt{G_i(\tau)+\epsilon}}\frac{\partial L}{\partial \theta_i}(\tau) \\ G_i(\tau) &= G_i(\tau-1) + \left(\frac{\partial L}{\partial \theta_i}(\tau)\right)^2. \end{aligned} \tag{3.44}$$

Recommending

Since the final layer is a softmax layer, we can interpret the scores for each item as a probability of it being the next item in the sequence. Thus, for predicting the next items of user u we feed his user-item interaction sequence into the CFRNN and rank the top n probabilities for each item in descending order. This means we only look at the final predictions instead of the predictions per time step. The items within this ordered list are the top n recommendations for user u.

3.5 Neural Collaborative Filtering

Combining the previously explained MF and MLP in a unique way is what X. He et al. (2017) describe in their NCF framework. The MF component operates similarly as explained in Section 3.2 but with a different approach to calculating the loss. Instead of a pairwise loss function, like BPR, their *Generalised Matrix Factorisation* (GMF) treats recommending as a binary classification problem. Simultaneously, the MLP component is used to model the non-linear user-item interaction function. The final output is then composed of their combined output, which we discuss in subsection 3.5.4. The full model is named *Neural Matrix Factorisation* (NeuMF), which is constructed under their proposed *Neural Network based Collaborative Filtering* (NCF) framework. Since this model is based on MF, we adopt the notation as specified in Table 3.1, which differs from the author's notation. The rest of this section will cover the NCF framework followed by a description of each component and finally their combination into NeuMF.

3.5.1 NCF Framework

In short, this framework allows the dot product of MF to be interchangeable with a DNN to map the user and item latent feature factors to prediction scores. As we are focused on comparing CF methods we take the binarised sparse vector representation of user-item interactions as the inputs. However, the authors state that different ways of modelling users and items can be adopted. The binarised user inputs pass through an embedding layer to create their latent feature vectors in a similar fashion as MF, the same goes for the item inputs. With these embeddings we formulate the mapping to a prediction as

$$\hat{y}_{ui} = f\left(p_u, q_i | p, q, \Theta_f\right), \tag{3.45}$$

where $p_u \in R^{\mathcal{M} \times \gamma}$ and $q_i \in R^{\mathcal{N} \times \gamma}$ denote the latent factor vector for users and items respectively. This is equivalent to the representation of p_u and q_i within MF (Section 3.2). Furthermore, Θ_f denotes the parameters of the interaction function f. The difference with standard MF is that the interaction function under NCF is defined as a multi-layer neural network, meaning it can be formulated as

$$f\left(p_u, q_i\right) = \phi_{out}\left(\phi_X\left(\ldots \phi_2\left(\phi_1\left(p_u, q_i\right)\right)\ldots\right)\right), \tag{3.46}$$

where ϕ_{out} and ϕ_X denote the mapping function for the output layer and the X-th neural collaborative filtering layer respectively.

Instead of using pairwise loss or classic pointwise loss we adopt binary cross-entropy loss, which is a special case of the previously explained categorical cross-entropy loss (subsection 3.4.4). Adopting a probabilistic approach for calculating $\hat{y}_{ui}$ fits both the binarised representation of implicit data, as well as the use of binary cross-entropy loss. To interpret the output as a probability we

constrain $\hat{y}_{ui}$ in the range of $[0, 1]$ using a sigmoid activation function (Equation 3.13) in the output layer ϕ_{out}. With this activation function in place we then define the likelihood function as:

$$p\left(\mathcal{I}^+, \mathcal{I}\backslash\mathcal{I}^+ | p, q, \Theta_f\right) = \prod_{(u,i)\in\mathcal{I}^+} \hat{y}_{ui} \prod_{(u,j)\in\mathcal{I}\backslash\mathcal{I}^+} \left(1 - \hat{y}_{uj}\right), \tag{3.47}$$

where $\mathcal{I}^+$ is the set of positive items and $\mathcal{I}\backslash\mathcal{I}^+$ is the set of negative items, which can be all or sampled from unobserved interactions per user. To obtain the objective function we take the negative logarithm of the likelihood:

$$\begin{aligned} L &= -\sum_{(u,i)\in\mathcal{I}^+} \log \hat{y}_{ui} - \sum_{(u,j)\in\mathcal{I}\backslash\mathcal{I}^+} \log\left(1 - \hat{y}_{uj}\right) \\ &= -\sum_{(u,i)\in\mathcal{I}^+\cup\mathcal{I}\backslash\mathcal{I}^+} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log\left(1 - \hat{y}_{ui}\right). \end{aligned} \tag{3.48}$$

To minimise L we perform mini-batch SGD as in the previously explained model (Section 3.4). As for the negative samples $\mathcal{I}\backslash\mathcal{I}^+$, we uniformly sample them from the unobserved interactions. This means we can control the ratio of negative instances we feed into the network and treat this as a hyperparameter.

3.5.2 Generalised Matrix Factorisation

Taking just the embedding layer above the input layer of NCF we obtain user and item latent feature vectors p_u and q_i. Now if we use only one NCF layer in which the mapping is simply the element-wise product of vectors we end up with the following form of standard MF:

$$\phi_1\left(p_u, q_i\right) = p_u \odot q_i, \tag{3.49}$$

where the element-wise product of vectors is denoted by $\odot$. Projecting this vector to the output layer according to the NCF framework results in

$$\hat{y}_{ui} = a_{out}\left(\mathbf{h}^\top \left(p_u \odot q_i\right)\right), \tag{3.50}$$

where $\mathbf{h}$ and a_{out} denote the edge weights and activation function of the output layer respectively. Taking $\mathbf{h}$ to be a uniform vector of 1 and for a_{out} an identity function, we recover the standard MF model under NCF.

Under NCF, X. He et al. (2017) define *Generalised Matrix Factorisation* (GMF) as Equation 3.50 where a_{out} is represented by the sigmoid function (Equation 3.13) and $\mathbf{h}$ is learned from the data using the binary cross-entropy loss and SGD.

In this case the updates of p_u and q_i happen in a similar fashion as with standard MF. However, now we start from the binary cross-entropy loss and calculate the gradient with respect to this loss. Next, using this gradient we update the corresponding embedding layers in proportion to a learning rate α. Lastly, the same L2 regularisation is used to lower overfitting on the training data.

3.5.3 Multilayer Perceptron

Instead of the straightforward approach of GMF, we first concatenate vectors p_u and q_i to be able to use a standard *Multi-layer Perceptron* (MLP) to learn the user item interactions. This allows for a large level of flexibility and non-linearity compared to the GMF model. Formally this MLP approach under the NCF framework can be defined as:

$$\begin{aligned} \mathbf{z}_1 &= \phi_1(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix} \\ \phi_2(\mathbf{z}_1) &= a_2\left(\mathbf{W}_2^\top \mathbf{z}_1 + \mathbf{b}_2\right) \\ &\dots \\ \phi_L(\mathbf{z}_{L-1}) &= a_L\left(\mathbf{W}_L^\top \mathbf{z}_{L-1} + \mathbf{b}_L\right) \\ \hat{y}_{ui} &= \sigma\left(\mathbf{h}^\top \phi_L(\mathbf{z}_{L-1})\right) \end{aligned} \tag{3.51}$$

where $\mathbf{W}_x$, $\mathbf{b}_x$ and a_x denote the weight matrix, bias vector, and x-th layer's activation function. The author's opt for *ReLU* as the activation function of the MLP layers for multiple reasons. Next to their empirical results, in which *ReLU* outperforms *tanh* and *sigmoid*, *ReLU* is proven to be non-saturated (Glorot, Bordes, & Bengio, 2011) and well-suited for sparse data. The tower structure is used as the MLP architecture, meaning we take half the size of the previous layer for the next layer. This is done such that the layers with less hidden units learn relatively more abstractive features of the users and item.

The steps specified in Equation 3.51, represent the forward propagation of the MLP. This representation is rather standardised and therefore not further explained in X. He et al. (2017). Hence, instead of focusing on the specifics of the back propagation in this work, we refer the curious reader to Algorithms 6.3 and 6.4 in subsection 6.5.4 of Goodfellow et al. (2016). The only addition to this standardised format is that the final partial derivative of $\mathbf{W}_2^\top$ with respect to the loss guides the update direction of the components of vectors p_u and q_i.

3.5.4 Neural Matrix Factorisation

The goal of creating GMF and MLP is that they can mutually reinforce each other when combined, this combination is defined as *Neural Matrix Factorisation* (NeuMF). To create NeuMF, both GMF and MLP components keep their original structure, each with their own embedding layer. The final output of NeuMF is then created by concatenating the last hidden layer of both components, as shown in Figure 3.8. Formally we define NeuMF as

$$\begin{aligned} \phi^{GMF} &= p_u^G \odot q_i^G \\ \phi^{MLP} &= a_L\left(\mathbf{W}_L^\top\left(a_{L-1}\left(\dots a_2\left(\mathbf{W}_2^\top \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + \mathbf{b}_2\right)\dots\right)\right) + \mathbf{b}_L\right) \\ \hat{y}_{ui} &= \sigma\left(\mathbf{h}^\top \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}\right), \end{aligned} \tag{3.52}$$

where p_u^G and q_i^G are the latent feature vectors of user u and item i in GMF respectively; and p_u^M and q_i^M similarly represent these vectors within the MLP's embedding layer. As stated by X. He et al. (2017), initialisation of the model

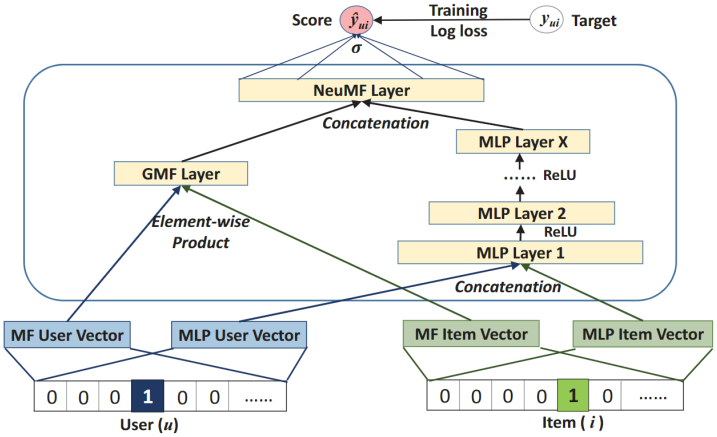


Figure 3.8: NeuMF architecture (X. He et al., 2017)

plays a key role in obtaining optimal performance. They note that initialising the weights of NeuMF with pre-trained weights from the individual components can lead to better convergence and performance of the combined model.

Note that the forward and backward passes through NeuMF equal the aforementioned propagation steps for GMF and MLP. The only difference is that there is an additional step before the sigmoid activation. Thus, we omit the derivation of forward and backward propagation of NeuMF.

Optimisation: Adam, SGD

While training GMF, MLP and NeuMF we optimise using *Adaptive Moment Estimation* (Adam) (Kingma & Ba, 2015). This method also adopts the individual learning rates per parameter like AdaGrad (3.4.4). However, it uses the first and second moments of the gradients to adapt the learning rate per parameter. More formally, using Kingma and Ba (2015), we can describe this

algorithm as:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \frac{\partial L}{\partial w}(t) \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot (\frac{\partial L}{\partial w}(t))^2 \\ \widehat{m}_t &= \frac{m_t}{(1-\beta_1^t)} \\ \widehat{v}_t &= \frac{v_t}{(1-\beta_2^t)} \\ \Delta\theta &= -\alpha \cdot \frac{\widehat{m}_t}{(\sqrt{\widehat{v}_t} + \epsilon)} \end{aligned} \quad [3.53]$$

Here β_1 and β_2 denote the decay rates for the first and second moment estimates respectively. Furthermore, m_t and v_t denote the first and second biased moment estimates at step t, in our case the steps are the mini-batches that we feed in to the algorithm. $\widehat{m}_t$ is the bias corrected form of m_t similarly for $\widehat{v}_t$ and v_t. Finally the difference in the parameters is defined as $\Delta\theta$ and α is the learning rate as defined before.

However, when initialising the weights of NeuMF using the pre-trained weights from GMF and MLP we are unable to keep track of the previously obtained momentum. Thus, in this case we use vanilla SGD as defined in Equation 3.43 to train NeuMF.

Recommending

The output of NeuMF is a probability based on the user-item pair, fed into the network. Probabilities closer to one can be interpreted as a larger personal preference towards an item than a probability closer to zero. Therefore, to obtain the top n recommended items for user u we feed user-item pairs into the network for all items and rank the resulting list of preference probabilities in descending order. The recommendations for user u are then defined as the top n items of this ranked list, similar to BPR.

Chapter 4

Experimental Setup

First, we elaborate on the structural differences between the Amazon Fashion dataset and the *MovieLens 1M data* (2003). To obtain additional insights on the difference in model performance we create a hybrid version of the Amazon Fashion and MovieLens datasets. With the structural analysis of the aforementioned datasets, we answer **SQ1**: What are the structural differences between fashion and movie data? Next, we adopt a similar training, validation and test split as Devooght and Bersini (2016) for assessing recommendation performance. In addition, we analyse and select two performance metrics for measuring recommendation performance, answering **SQ2**: How to measure model performance, and which metric is most suitable for our research? Finally, we provide a detailed description of the setup for BPR, CFRNN and NeuMF.

4.1 Data

The 5-core Amazon Clothing Shoes and Jewellery dataset (Ni et al., 2019; *Amazon Review data*, 2018) is a review dataset of a subset of products sold by e-commerce giant Amazon. The item categories are similar to the dataset *YGroup* (Y) is facing for the application of the models explored in this work. As mentioned in Chapter 1, we ignore the ratings and consider each rating to be a purchase, resulting in a purchase history dataset. This means the only values utilised by the algorithms are *user id*, *item id* and *datetime*. The combination of these three features stands for an interaction per *user id* on the specified *datetime*, with the *item id*. Due to the full dataset consisting of 11 285 464 reviews and limited resources we conduct this research on two different subsets of the full Amazon data. Note that before taking a subset of the Amazon dataset, many non-fashion items that were still present are removed, e.g., wallpapers and books. As for the MovieLens 1M dataset, this is already a subset of the MovieLens 25M dataset (*MovieLens 25M data*, 2019). Both characteristics of the full Amazon and MovieLens dataset can be found in Appendix A.

This section elaborates on the specifications of the Amazon, MovieLens and Amazon MovieLens hybrid datasets and their structural differences. Furthermore, we explain the reasoning behind the training, validation and test split used to assess model performance. Finally, we motivate each model's choice of initialisation and hyperparameters.

4.1.1 Amazon 20k Users

The first subset obtained from the 5-core Amazon Clothing Shoes and Jewellery dataset is the Amazon 20k Users subset. Since this research utilises CF algorithms it is important for each user to have some interaction history in the data, meaning each user needs a minimum number of interactions. Similar to the 5-core dataset we take a minimum of five user-item interactions per user to create the Amazon 20k Users subset. Naturally, when sampling users from the full data the number of ratings (interactions) per item decreases. The specifications of this subset can be found in Table 4.1. We observe a large number

Table 4.1: Characteristics of Amazon 20k Users subset

General Statistics	Value
Total Interactions	180 809
Total Users	20 000
Total Items	90 395
Sparseness	99.999%
Average Rating	4.28/5
Interactions Per User	
Average	9.04
Median	7.0
Interactions Per Item	
Average	2
Median	1

of items compared to the number of users, together with an average number of ratings per user of 9.04. This combination produces the low number of ratings per item in Figure 4.1. Furthermore the average rating for all items is heavily left-skewed, meaning most items are highly rated between 4 and 5 on a scale from 1-5. This skewness reinforces the implicit feedback assumption that an interaction implies a user's preference for the rated item.

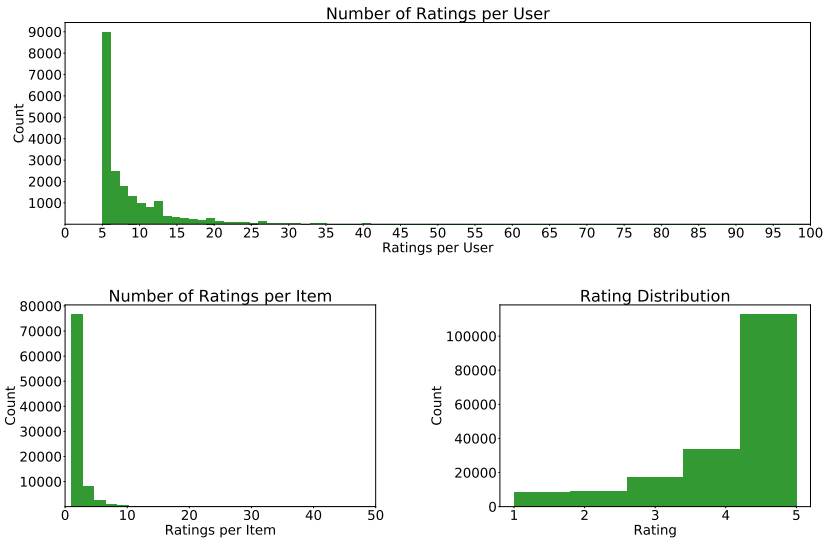


Figure 4.1: Distributions of the number of ratings, the number of ratings per item and the rating scores for the Amazon 20k users dataset (see Appendix A.2 for a long-tailed focused representation)

4.1.2 MovieLens 1M

Within the literature reviewed in Chapter 2, the MovieLens 1M ratings dataset is often used to assess performance of recommender algorithms. The characteristics of this dataset are shown in Table 4.2. With 3 706 movies and 1 000 209around one million ratings we observe a minimum of 20 ratings per user with a long tail of larger ratings (Figure 4.2). A difference with the Amazon 20k users dataset,

Table 4.2: Characteristics of MovieLens 1M

General Statistics	Value
Total Interactions	1 000 209
Total Users	6 040
Total Items	3 706
Sparseness	99.9553%
Average Rating	3.58/5
Interactions Per User	
Average	165.6
Median	96.0
Interactions Per Item	
Average	269.89
Median	123.5

however, is the fact that the rating distribution is less left-skewed. Here, most items are actually rated between 3 and 4 out of 5.

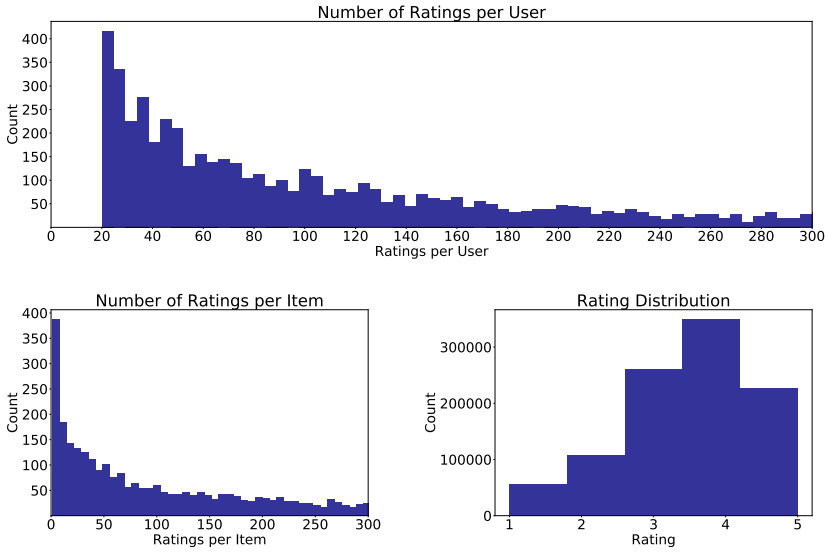


Figure 4.2: Distributions of the number of ratings, the number of ratings per item and the rating scores for the MovieLens 1M dataset (see Appendix A.2 for a long-tailed focused representation)

4.1.3 Amazon like MovieLens 1M

With the substantial gap in items, users and number of ratings per user between the previously introduced datasets, we propose a version of the full Amazon dataset which more closely resembles the structure of the MovieLens 1M dataset. The goal of this new subset, named Am-like-ML, is to observe the difference in performance of the algorithms on a dataset that contains characteristics of both Amazon Fashion and MovieLens, as shown in Table 4.3. The Am-like-ML subset is created by taking an equal amount of users as observed in the MovieLens 1M subset, where all users have a minimum of 20 interactions. As shown in Figure 4.3, some of the structural differences remain, such as the low number of ratings per item and the heavily left-skewed distribution of ratings.

Table 4.3: Characteristics of Am-like-ML subset

General Statistics	Value
Total Interactions	178 794
Total Users	6 040
Total Items	87 290
Sparseness	99.9996%
Average Rating	4.29/5
Interactions Per User	
Average	29.6
Median	25.0
Interactions Per Item	
Average	2.05
Median	1.0

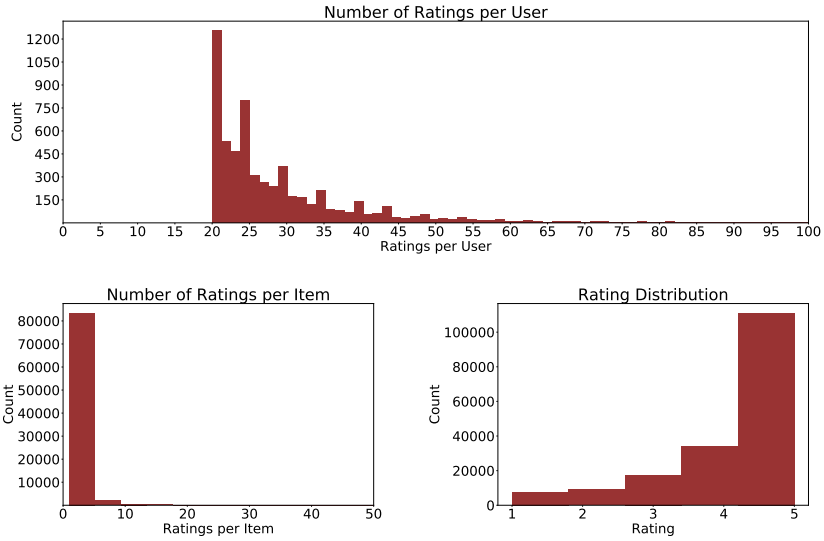


Figure 4.3: Distributions of the number of ratings per user, the number of ratings per item and the rating scores for the Am-like-ML dataset (see Appendix A.2 for a long-tailed focused representation)

4.1.4 Structural Differences

Putting the highlighted differences in the previous subsections together, we obtain Figure 4.4. The number of users and the minimum number of interactions per user is now equal for Am-like-ML and MovieLens 1M. However, the number of ratings of Am-like-ML is similar to that of Amazon 20k Users. A structural difference that remains unchanged is the total number of items for the Am-like-

ML subset, which is more representative of fashion than of movies (see full data characteristics in Appendix A). Besides the total number of items, the rating distribution also retains the structure found in Amazon 20k Users as shown in Figures 4.1, 4.2 and 4.3.

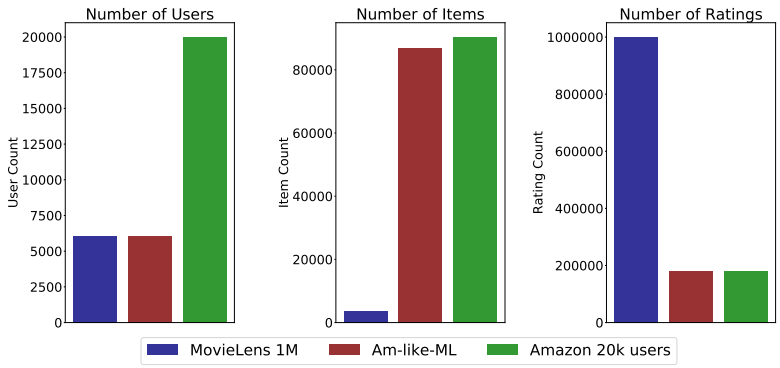


Figure 4.4: Comparison of the number of users, items and ratings for Amazon 20K Users, MovieLens 1M and Am-like-ML

4.1.5 Training, Validation and Test Split

Common practice in the literature is to split the data according to the leave-one-out strategy. Here, the last item a user interacted with (chronologically) is removed from the dataset to serve as an item for validation or testing. This implies that the rest of the data can be used in training. Since the algorithms adopted in this work differ in the way they utilise the available data, we propose a modified version of this split.

We propose taking a subset of users using the leave-one-out strategy to create the test set, this way we retain as many items per user in the training set as possible. For the validation set, we exclude the users already present in the test set. In addition, we need to take the differences between MF and sequence prediction-based models into account. For BPR and NeuMF we need user-item interaction history to update the latent factor vectors while training, as these are individually defined per user and per item. The CFRNN on the other hand, needs a user-item interactions sequence as input to predict the next items in the sequence. Therefore, we cannot utilise the item sequences of the test users to train on. Devooght and Bersini (2016) split the test sequences in half, using the first half as input and evaluate performance on the difference in predicted items and the second half of the true sequence. We propose a similar split but instead of predicting the second half of their sequences, we predict only the final item, meaning we feed all items except the held-out item of the test users into the algorithm to generate the top n recommendations. Thus, for CFRNN

the training, validation and test set users will not overlap, while for BPR and NeuMF the users in the training set overlap with both the validation users and the test users. One disadvantage of this split is the fact that the MF-based algorithms observe more user-item interactions than the CFRNN during training (Table 4.4). However, this unfairness is inevitable when comparing these types of algorithms, as also pointed out by Devooght and Bersini (2016). The characteristics per dataset in terms of number of users for the training, validation and test sets is shown in Table 4.5.

Table 4.4: Difference in training and test split between MF-based and CFRNN algorithms for a single test user (chronologically ordered)

Algorithm	Training set*	Needed for predicting	Test set
MF-based	$i_1, i_2, ..., i_{k-1}$	-	i_k
CFRNN	-	$i_1, i_2, ..., i_{k-1}$	i_k

$^\star i_1$ represents the first item id in the user's sequence, k denotes the length of the user's item sequence

Table 4.5: Number of users in the training, validation and test splits for CFRNN and MF-based algorithms for Amazon 20k Users, MovieLens 1M and Am-Like-ML

CFRNN	Am 20k Users	MovieLens 1M	Am-like-ML
train	18 500	4 540	4 540
test	1 000	1 000	1 000
validation	500	500	500
MF-Based			
train	20 000	6 040	6 040
test	1 000	1 000	1 000
validation	500	500	500

4.2 Performance Metrics

Before explaining the modelling setup, we answer the second sub-question **SQ2**: how to measure model performance? Since we are classifying which items to recommend, classification performance metrics are considered. Furthermore, note that each algorithm uses a ranking of the final item scores per user. Therefore, the position of the held-out test item among the other predicted items will provide more insight into model performance. Thus, this section elaborates on the choice of classification- and ranking metrics to measure recommendation performance.

Recommending

For recommending with BPR, CFRNN and NeuMF we follow the procedures as specified in 3.3.1, 3.4.4 and 3.5.4 respectively. Note that the items already present in the user's interaction history are not considered when recommending. This is enforced by setting the score of past interaction items as low as possible during the ranking of the item scores.

4.2.1 Classification: Recall@n

This classification boils down to predicting which items are interesting to the user and which items are not. Thus this classification results in the following confusion matrix shown in Table 4.6. As one of the main objectives of recommendation systems is to narrow down the items which appeal to a specific user, we evaluate the performance at n. This implies that whenever the true next item (held-out test item) of the user is among the predicted n items for that user, we observe a True Positive.

Table 4.6: Confusion Matrix for Recommendation Systems

	Interacted with	Not interacted with
Recommended	*True Positives* (TP)	*False Positives* (FP)
Not Recommended	*False Negatives* (FN)	*True Negatives* (TN)

Recall measures what proportion of actual positives is correctly classified:

$$Recall = \frac{TP}{TP + FN} \quad [4.1]$$

High recall signifies we captured many positives from all positives in the data. Since we are recommending a subset of all items per user, we measure recall within this subset. This results in the $recall@n$ metric, where n is the length of the recommended subset. Since we have one held-out item in the test set, we observe a recall of 1 per user if this item is observed in the top n recommendations and 0 if it is not. We define the final $recall@n$ metric as the average recall calculated over all test users. To obtain more insight in model performance we obtain $recall@n$ for $n \in \{1, 5, 10, 15, 20\}$. Note that a $recall@1$ equal to 1.0 represents a perfect classification score and will automatically set $recall@n$ for any $n > 0$ equal to 1. With a $recall@n$ of 0.0, no held-out item is correctly classified within the item subset of length n for all users involved.

4.2.2 Ranking: NDCG@n

With a $recall@1$ score of 1.0 we not only observe a perfect classification, but also a perfect ranking. However, for $recall@n$ where $n > 1$, this score does not provide insight in the exact ranking of the held-out item anymore. Thus, the

recall@n score only confirms if the held-out test item is in the top n recommendations, it does not take the position of this item within the top n into account. Therefore, we include a ranking metric which provides additional insights in model performance. For the ranking problem we can describe our top n recommendations as:

$$recommendations = i_1^r, i_2^r, \ldots, i_n^r, \quad (i \in \mathcal{I},\ r \in \{0, 1\}), \tag{4.2}$$

where r denotes the relevance of the item. Since we hold out one item per user for the test set, we have $\mathcal{M} - 1$ items (see General notation in Table 3.1) where $r = 0$ and a single item where $r = 1$. The rank of this i^1 among the top n items is what needs to be measured. We adopt the popular *Normalised Discounted Cumulative Gain* (NDCG) as our ranking metric to evaluate ranking performance.

NDCG is build upon the basic concept of *Cumulative Gain* (CG) which is defined as the sum of all the relevance scores in a given set:

$$CG = \sum_{j=1}^{n} \mathrm{r}_j. \tag{4.3}$$

In our case CG equals hitcount, as there is only one relevant item with relevance score 1. Thus, to take the position of this one item into account we use *Discounted Cumulative Gain* (DCG). Discounting the relevance score by dividing it by the *log* of the corresponding position allows us to take the position of each item into account:

$$DCG = \sum_{j=1}^{n} \frac{\mathrm{r}_j}{\log_2(j+1)}. \tag{4.4}$$

For completeness, we normalise DCG to arrive at NDCG. This step ensures recommendations of various sizes are measured in the same way. For this we divide DCG by the ideal order (iDCG):

$$NDCG = \frac{DCG}{iDCG}. \tag{4.5}$$

Finally, to calculate the total ranking score we average the NDCG at cut-off point n for all users in the test set, resulting in $NDCG@n$. A perfect NDCG@n of 1.0 means all held-out items are ranked first in the top n recommendations.

4.3 Bayesian Personalised Ranking

The initialisation settings of BPR and its hyperparameters are shown in Table 4.7 and Table 4.8 respectively. The parameters that differ per dataset have been found using a grid search per combination of algorithm and dataset of which the results can be found in Appendix B. As already mentioned in Section 3.3, the samples consist of a user, a positive item and a negative item, randomly

Table 4.7: Initialisation of BPR

Component	Initialisation	Parameters*
user latent factor matrix p	random normal	$\mu = 0,\ \sigma = 0.1$
item latent factor matrix q	random normal	$\mu = 0,\ \sigma = 0.1$

* μ and σ represent the mean and standard deviation of the normal distribution respectively.

Table 4.8: Hyperparameters used in BPR for each dataset

Parameters	Amazon 20K Users	MovieLens 1M	Am-Like-ML
γ	8	8	8
Epochs	25	25	25
α	0.05	0.05	0.08
ρ	1.05	1.05	1.05
σ	0.55	0.55	0.55
λ_p	0.1	0.001	0.1
λ_q	0.1	0.001	0.1
Sample Size	89 654	99 870	141 918
Sample% of Interactions	50%	10%	80%

sampled from the training set. Therefore the Grid Search and final results are based on the same samples. Since the number of interactions is considerably large and many users need to be considered, we use 25 epochs. The sampling ratio is an important parameter as we limit the algorithm to 25 epochs, the correct sampling ratio decides how many triples are observed per epoch. The difference in sampling ratio between MovieLens 1M and Amazon 20k users can be explained by the difference in the number of users and the number of interactions per user.

We use 8 as the dimension of the latent feature vectors (γ) as this is commonly used as the smallest dimension within the literature. With this minimum dimension we reduce computing time while still obtaining adequate performance. Using a larger value for γ can result in a better abstract representation of users and items, leading to better recommendation performance. As optimisation of the utilised models is not the objective of this work, we leave larger values of γ out of scope.

The learning rate and regularisation parameters are chosen based on the aforementioned grid search (Appendix B) and its results on the validation set. More specifically we utilise the parameters that achieved the largest validation recall@10 after 25 epochs.

Since the Bold-Driver heuristic adapts the learning rate based on the loss value,

we keep ρ and σ constant but vary the learning rate α. Thus, we observe different initial learning rates during the grid search. Common values for the Bold-Driver approach are around 1 for ρ and around 0.5 for σ (Shepherd, 2012).

The difference in regularisation can be explained by the difference in the number of items between the MovieLens 1M dataset and the others. Within Amazon 20K Users and Am-like-ML more items need to be considered with less ratings per item, meaning one update to the latent features of an item has more impact on the results than for the MovieLens 1M updates. Thus, using more L2 regularisation for the latent feature factors of the datasets with less ratings per item and less ratings per user could limit overfitting as the updates have a smaller impact on the latent feature factors. Note that during the grid search we kept $\lambda_p = \lambda_q$, however, varying these values individually per run could result in greater recommendation performance and is left for future research.

4.4 Collaborative Filtering with Recurrent Neural Networks

The CFRNN contains four different layers, of which three have to be initialised. Each initialisation approach and their parameters are shown in Table 4.9. Initialisation plays an important role within RNNs, as already mentioned in Section 3.4, vanilla RNN units suffer from exploding or vanishing gradients. Certain weight initialisation in a DNN layer can bring about unstable gradients because of the combined variance of the layer's input units. Methods to restrict the initial variance during initialisation include Glorot- (Glorot & Bengio, 2010) and Lecun initialisation (LeCun, Bottou, Orr, & Müller, 1998). The

Table 4.9: Initialisation of CFRNN

Component	Initialisation	Parameters*
Embedding layer	random uniform	$b = [-0.05, 0.05]$
Recurrent LSTM layer	Glorot uniform	$b = [-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$
Dense Layer	Glorot uniform	$b = [-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$

* b represents the boundaries for uniformly drawing the weight and is a preset parameter for the glorot uniform initialisation based on Glorot and Bengio (2010). fan_{in} denotes the number of input units in the weight tensor, fan_{out} the number of output units.

hyperparameters used for the CFRNN algorithm are shown in Table 4.10 and are also obtained using a grid search per dataset (Appendix B). Note that for the Amazon 20K Users we observed overall poor performance in the grid search results. This performance gap between the Amazon 20K Users and the other datasets can be caused by the relatively short user-item interaction sequences

within Amazon 20K Users. For the parameters found for MovieLens 1M we experimented with less hyperparameters as Devooght and Bersini (2016) already explored various combinations of this model with this exact dataset. The Mask value is taken to be $\mathcal{M}$ to make sure we do not mask items that are present in user sequences, meaning item ids range from 0 to $\mathcal{M} - 1$.

Table 4.10: Hyperparameters used in CFRNN for each dataset

Parameters	Amazon 20K Users	MovieLens 1M	Am-Like-ML
δ	0.2	0.01	0.01
RNN Units	20	20	50
Epochs	20	100	20
α	0.1	0.2	0.1
Batch Size	32	16	64
Max Sequence Length	20	30	30
Embedding Dimension	100	100	100
Mask Value	90 395	3 706	87 290

4.5 Neural Collaborative Filtering

The initialisation of the user and item latent factor matrices is performed in a similar fashion as for BPR, shown in Table 4.11. In addition, the MLP layers on top of p^M and q^M together with the final layer use Glorot and Lecun uniform initialisation. Similar to the other algorithms, we used a grid search per dataset to find optimal parameters for NeuMF. For the same reason as keeping $\gamma = 8$ for BPR and in terms of keeping the comparison as fair as possible, we use $\gamma = 8$ for the GMF component of NeuMF. Since the final dense layer of this

Table 4.11: Initialisation of NeuMF

Component*	Initialisation	Parameters**
p^G	random normal	$\mu = 0,\ \sigma = 0.05$
q^G	random normal	$\mu = 0,\ \sigma = 0.05$
p^M	random normal	$\mu = 0,\ \sigma = 0.05$
q^M	random normal	$\mu = 0,\ \sigma = 0.05$
MLP layers	Glorot uniform	$b = [-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$
Final dense layer	Lecun uniform	$b = [-\sqrt{\frac{3}{fan_{in}}}, \sqrt{\frac{3}{fan_{in}}}]$

* p^G and q^G stand for user and item latent factor matrices of GMF respectively. p^M and q^M denote user and item latent factor matrices of MLP respectively.
** μ and σ represent the mean and standard deviation of the normal distribution respectively. b represents the boundaries for uniformly drawing the weight and is a preset parameter based on Glorot and Bengio (2010) and LeCun et al. (1998). fan_{out} denotes the number of output units, fan_{in} denotes the number of input units in the weight tensor.

algorithm is a concatenation of GMF and MLP we need the same final dimension. Consequently, keeping $\gamma = 8$ constrains the MLP layers to follow a tower structure of 16, 32, 16, 8, where the first 16 is the combination of user and item embeddings. Furthermore, within the grid search we utilise 4 and 8 negatives per input sample. However, as the authors have tested any number of negatives up to 10 for MovieLens 1M we restricted the corresponding grid search to only use 4 negatives as this is the optimal result of their extensive testing.

Due to empirical evidence of rapid loss and recall@10 conversion in X. He et al. (2017) and high computational needs we keep the number of epochs to 20. Note that the previously mentioned research found empirical evidence for improved performance of NeuMF when the weights are initialised by pre-trained GMF and MLP components. We do not incorporate pre-training for weight initialisation because this introduces more stochastic components which have to be taken into account when obtaining final performance metrics and testing for statistically significant results. In other words, training both components of NeuMF before training NeuMF itself leads to a significant increase in computing

time needed to complete training. In addition, the empirical evidence shown in Table 2 of X. He et al. (2017) exhibits no concluding evidence of performance improvement when using pre-trained components and a γ equal to 8. The regu-

Table 4.12: Hyperparameters used in NeuMF for each dataset

Parameters	Amazon 20K Users	MovieLens 1M	Am-Like-ML
γ	8	8	8
Layers	16, 32, 16, 8	16, 32, 16, 8	16, 32, 16, 8
Epochs	20	20	20
Regularisation GMF	1e-06, 1e-06	0,0	1e-05, 1e-05
Regularisation MLP	0.0001, 0.0001, 0.0001, 0.0001	0, 0, 0, 0	0.0001, 0.0001, 0.0001, 0.0001
α	0.0001	0.00005	0.00005
Batch Size	512	512	512
#Negatives	4	4	8
Sample Size	889 045	4 993 545	1 584 108
Sample% of Interactions	500%	500%	900%

larisation components seem to be based on the same arguments provided for the regularisation of BPR. Here, less interactions per item seems to cause relatively more regularisation for both the GMF component and the MLP component.

Chapter 5

Experimental Results

This Chapter showcases the results and comparison of *Bayesian Personalised Ranking* (BPR), *Collaborative Filtering with Recurrent Neural Networks* (CFRNN) and *Neural Matrix Factorisation* (NeuMF) for the Amazon 20K Users, Movielens 1M and Am-like-ML datasets. First, we elaborate on the implementation setup in Section 5.1. Next, sections 5.2, 5.3 and 5.4 show the loss together with the validation recall@10 and NDCG@10 on each dataset, for each algorithm respectively. After the individual results we show a comparison for all algorithms per dataset in terms of recall@n and NDCG@n for $n \in \{1, 5, 10, 15, 20\}$ on their respective test set. Finally, we present the p-values of the Paired T-Test for testing the difference in average recall@n and average NDCG@n between algorithms for every dataset.

5.1 Implementation Setup

As all algorithms presented in Chapter 3 involve stochastic optimisation and random initialisation, we perform 30 runs per algorithm on each dataset, where a run is defined as training and testing. For these runs we log the following values:

- Loss per epoch
- Validation recall@10 per epoch
- Validation NDCG@10 per epoch
- Recall@n, $n \in \{1, 5, 10, 15, 20\}$ on the test set as specified in 4.1.5
- NDCG@n, $n \in \{1, 5, 10, 15, 20\}$ on the test set as specified in 4.1.5

In the graphical representation of the results, we show the average and standard deviation of these logged results over the 30 runs. The results obtained for comparison then consist of 30 recall@n and NDCG@n scores for all ranks considered. In this work, an algorithm only outperforms another algorithm when

the average recall@n and average NDCG@n are proven to be statistically different from one another and both scores of one algorithm are above those of the other, for all n.

Statistical Testing

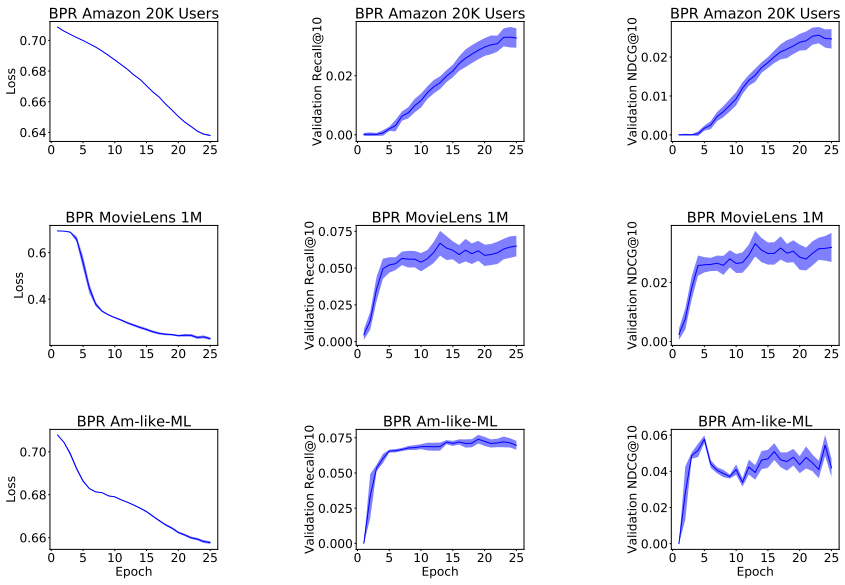
To test whether the difference in means between the results per algorithm is statistically significant, we utilise a Paired T-Test. For this test, we consider a significance level of 0.01. Since the 30 results are realised using the same test set for each algorithm, we cannot assume independence of the results when comparing the observations in these samples. Therefore, we test the difference in means between the samples using a Paired T-Test. With 30 observations per sample, we assume normality within the sample, based on the Central Limit Theorem. The corresponding null and alternative hypothesis are defined as

$$\begin{aligned} H_0 &: \mu_1 - \mu_2 = 0 \\ H_A &: \mu_1 - \mu_2 \neq 0, \end{aligned} \tag{5.1}$$

where μ_1 is the mean of the resulting 30 observations of one algorithm and μ_2 represents the same value for the other algorithm involved in the comparison. For the difference between means of the two populations to be considered significant, we need the p-value to be below a significance level of 0.01.

5.2 Bayesian Personalised Ranking

Figure 5.1 shows the average loss, validation recall@10 and validation NDCG@10 during training for 30 runs of BPR on each dataset. The standard deviation of the result sample is shown as the similarly coloured area around the mean. We observe monotonically decreasing loss functions for all datasets in Figure 5.1a. The validation metrics in Figure 5.1b and 5.1c both show upward trends with signs of convergence for both MovieLens 1M and Am-like-ML.

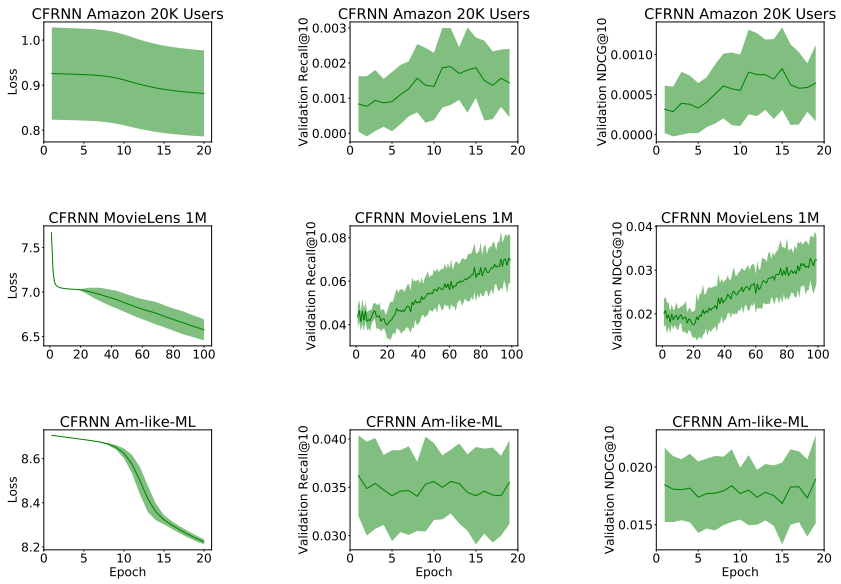


(a) Loss (b) Validation Recall@10 (c) Validation NDCG@10

Figure 5.1: Average and standard deviation per epoch of the Loss (a), Validation Recall@10 (b) and Validation NDCG@10 (c) of BPR on MovieLens 1M, AM-like-ML and Amazon 20K Users.

5.3 Collaborative Filtering with Recurrent Neural Networks

Figure 5.2 follows the same structure as Figure 5.1, but the results are obtained using the CFRNN algorithm. We showcase the average training loss and average validation metrics obtained using the parameters specified in Section 4.4 for 30 runs each. In Figure 5.2a we observe a clear downward trend with little standard deviation for MovieLens 1M and Am-like-ML; however, Amazon 20K Users shows a different pattern with a relatively large standard deviation. Note that the number of epochs for MovieLens 1M is set to 100 as a result of the aforementioned grid search. Only the validation metrics for MovieLens 1M show a clear upward trend.

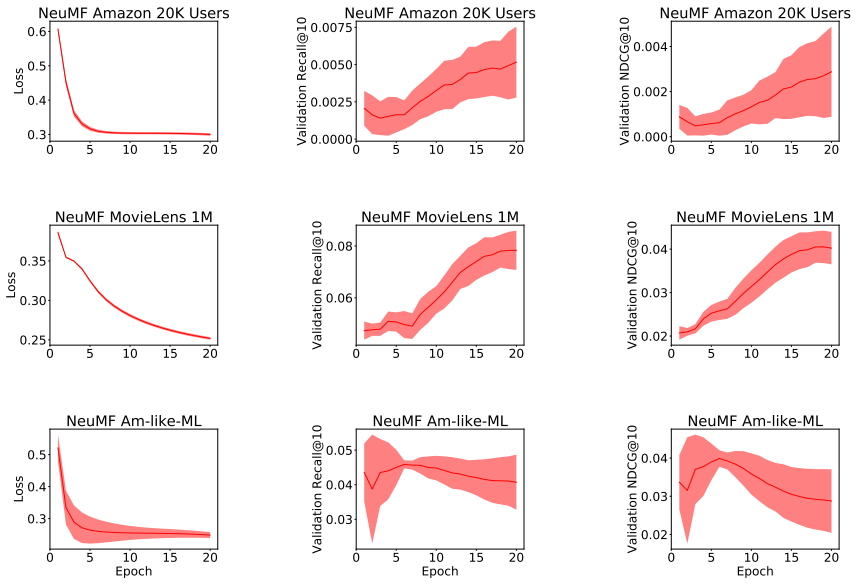


(a) Loss (b) Validation Recall@10 (c) Validation NDCG@10

Figure 5.2: Average and standard deviation per epoch of the Loss (a), Validation Recall@10 (b) and Validation NDCG@10 (c) of BPR on MovieLens 1M, AM-like-ML and Amazon 20K Users.

5.4 Neural Matrix Factorisation

Here we observe the average loss and average validation metrics of NeuMF on the three different datasets, obtained over 30 runs (Figure 5.3). First of all, every loss function is monotonically decreasing, as observed in Figure 5.3a. Both validation metrics of Amazon 20K Users and Movielens 1M show an upward trend (Figure 5.3b, 5.3c). As for the Am-like-ML dataset we observe a relatively large standard deviation for the average training loss and no clear trend in the validation metrics.

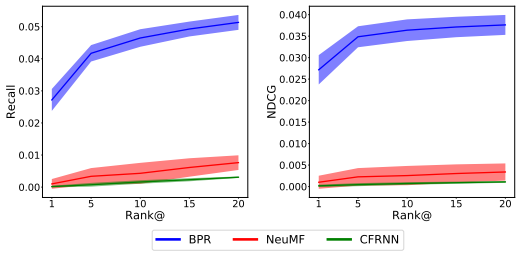


(a) Loss (b) Validation Recall@10 (c) Validation NDCG@10

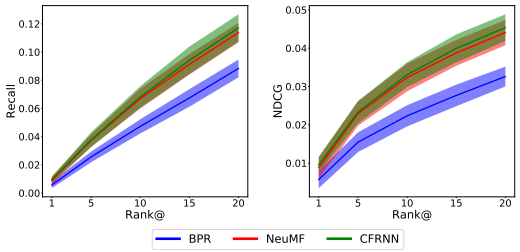
Figure 5.3: Average and standard deviation per epoch of the Loss (a), Validation Recall@10 (b) and Validation NDCG@10 (c) of BPR on MovieLens 1M, AM-like-ML and Amazon 20K Users.

5.5 Comparison

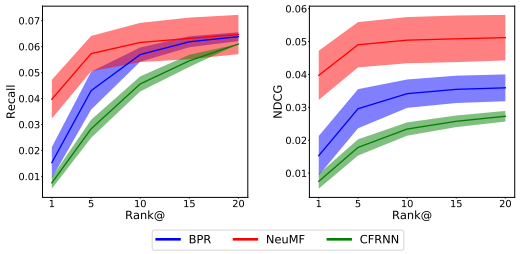
Figure 5.4 showcases the comparison of the average test set results of BPR, CFRNN and NeuMF for the Amazon 20K Users, MovieLens 1M and Am-like-ML datasets in terms of recall@n and NDCG@n. For each algorithm, we obtain the recall@n and NDCG@n performance metrics for $n \in \{1, 5, 10, 15, 20\}$ (rank@n) on the held-out test set. Averaging these metrics over the 30 runs results in the plots below, where each line represents the average performance metric for a single model. The coloured area around each line represents the standard deviation of the 30 results. Finally we showcase the Paired T-Test p-values tables for each algorithm comparison on each dataset for both reacll@n (Table 5.1) and Table 5.2).



(a) Amazon 20K Users



(b) MovieLens 1M



(c) Am-like-ML

Figure 5.4: Average Recall@n and NDCG@n of BPR, CFRNN and NeuMF over 30 runs per algorithm, together with their standard deviation for Amazon 20K Users (5.4a), MovieLens 1M (5.4b) and AM-like-ML (5.4c)

Table 5.1: p-values of the Paired T-Test for recall@n for all algorithm combinations per dataset, all p-values rounded to 8 decimals. Bold algorithm names indicate which algorithm dominates the other graphically (Figure 5.4) with all mean comparisons per Rank@ being significantly different (p-value below 0.01).

Amazon 20K Users			
Rank@	**NEUMF** vs. CFRNN	**BPR** vs. CFRNN	**BPR** vs. NEUMF
1	0.00864651**	0.0***	0.0***
5	2.16e-06***	0.0***	0.0***
10	7.468e-05***	0.0***	0.0***
15	2e-08***	0.0***	0.0***
20	0.0***	0.0***	0.0***
MovieLens 1M			
Rank@	NEUMF vs. CFRNN	BPR vs. **CFRNN**	BPR vs. **NEUMF**
1	0.22680662	2.2e-07***	0.00021738***
5	0.8130222	0.0***	0.0***
10	0.55757215	0.0***	0.0***
15	0.21582326	0.0***	0.0***
20	0.17299948	0.0***	0.0***
Am-like-ML			
Rank@	NEUMF vs. CFRNN	**BPR** vs. CFRNN	BPR vs. NEUMF
1	0.0***	1.6e-07***	0.0***
5	0.0***	0.0***	1e-08***
10	0.0***	0.0***	0.00306358**
15	1.71e-06***	0.0***	0.3858058
20	0.01282656*	0.0***	0.56555254

* p value below significance level 0.05
** p value below significance level 0.01
*** p value below significance level 0.001

Table 5.2: p-values of the Paired T-Test for NDCG@n for all algorithm combinations per dataset, all p-values rounded to 8 decimals. Bold algorithm names indicate which algorithm dominates the other graphically (Figure 5.4) with all Rank@ mean comparisons being significantly different (p-value below 0.01).

Amazon 20K Users			
Rank@	**NEUMF** vs. CFRNN	**BPR** vs. CFRNN	**BPR** vs. NEUMF
1	0.00864651**	0.0***	0.0***
5	2.967e-05***	0.0***	0.0***
10	9.133e-05***	0.0***	0.0***
15	4.58e-06***	0.0***	0.0***
20	5.4e-07***	0.0***	0.0***
MovieLens 1M			
Rank@	NEUMF vs. CFRNN	BPR vs. **CFRNN**	BPR vs. **NEUMF**
1	0.22680662	2.2e-07***	0.00021738***
5	0.64151823	0.0***	0.0***
10	0.4566743	0.0***	0.0***
15	0.24818406	0.0***	0.0***
20	0.20642687	0.0***	0.0***
Am-like-ML			
Rank@	**NEUMF** vs. CFRNN	**BPR** vs. CFRNN	BPR vs. **NEUMF**
1	0.0***	1.6e-07***	0.0***
5	0.0***	0.0***	0.0***
10	0.0***	0.0***	0.0***
15	0.0***	0.0***	0.0***
20	0.0***	0.0***	0.0***

* p value below significance level 0.05
** p value below significance level 0.01
*** p value below significance level 0.001

Chapter 6

Analysis and Discussion

The objective of this chapter is to provide an in-depth analysis of the experimental results as presented in Chapter 5 and answer **SQ3**: How do the structural differences between the datasets affect model performance?

In addition, we discuss the setup and shortcomings of this work. This chapter is structured similar to the previous chapter, meaning we focus on the algorithm setups and their training procedures first. Next, we evaluate the comparison and discuss the impact of each dataset on the performance comparison of the algorithms.

6.1 Bayesian Personalised Ranking

This section uses the training loss and metrics as shown in Figure 5.1, together with the formal notation and algorithm description provided in Section 3.1 and Section 3.3 respectively. First we consider the setup of BPR, followed by an evaluation of the training loss and metrics.

Setup

BPR assumes that if user u interacted with item i (positive item), it is preferred over non-interacted item j (negative item). Therefore, the algorithm's latent factor matrices p and q are updated based on the assumption that i should have a larger preference score than j. Another consequence of this formulation of the recommendation problem is the way data is considered during training. We sampled uij triples from the total number of user-item interactions (positive items) to train the model. Since these samples were created at random, we expect each sample to be different, meaning the model observed different samples at each epoch. As the test set consisted of one held-out item for a subset of users we note that different samples can lead to different results. For a more complete comparison, one could consider creating a multitude of

samples on which a number of BPR algorithms are trained and tested on different test sets. However, such an approach, together with testing for statistically significant results would require a substantial amount of time and resources.

Furthermore, during this sampling procedure, one could observe a bias towards users and items with the majority of the interactions. Especially in datasets similar to MovieLens 1M, where some users and items account for more than 300 interactions, while the majority of users and items have less than 100 interactions (Figure 4.2). One can counteract this sampling bias by sampling based on popularity in which popularity represents the number of interactions of a user or item. The lower an item's popularity, the more likely that item is selected during sampling.

As mentioned in Rendle et al. (2012), the random sampling also prevents consecutive updates on the same user-item pairs. Following the standard user or item order while sampling can lead to updating the same positive items every iteration, while also updating the negative items. Since preference towards i has to be larger than j, consecutive updates of j can lead to poor convergence of SGD.

As already mentioned in the experimental setup, we utilise a γ equal to eight to limit computational needs. However, a larger value of γ is often associated with better performance as shown in the results of Rendle et al. (2012) and X. He et al. (2017).

Training Loss and Metrics

The loss shown in Figure 5.1a is monotonically decreasing for all datasets while the validation metrics show an increasing trend. This indicates the algorithm learned to recommend during the training process. The structural differences between the datasets reveal themselves in the different shapes of the loss functions for each dataset. Considering the samples used for MovieLens 1M we expect many samples to contain the same items and users as there were a limited number to choose from. Or more specifically, with 6 040 users, 3 706 items and a sample size of 99 870, BPR performed frequent updates for each user and each item for MovieLens 1M. This indicates that the algorithm learns more about each user's preferences per epoch for MovieLens 1M than for the other datasets.

With less interactions per user and per item, the loss functions of the other datasets follow the reverse reasoning as for Movielens 1M. The most extreme case, Amazon 20K Users, reinforces this theory as there are even more users, all with relatively fewer interactions than the other datasets. Different sample sizes, learning rates and number of epochs might change this perspective on the shape of the loss curves.

For both the recall@10 and NDCG@10 curves we observe a rapid increase to a more steady-state reached around the fifth epoch for MovieLens 1M and Am-like-ML. However, this sharp increase developed at a slower pace for the Amazon 20K Users dataset. We believe this relatively slow increase in the validation metrics was due to the large number of users with little interactions. With a total of 180 809 interactions, 20 000 user and a sample size of 89 654 triples, not all users have to be considered per epoch, let alone all items. A larger number of epochs in the case of Amazon 20K Users could provide more insights into the convergence of these metrics.

6.2 Collaborative Filtering with Recurrent Neural Networks

Following the same structure as the previous section, we use the information from the Experimental Setup (Chapter 4), loss and metrics as shown in Figure 5.2, formal notation from Section 3.1 and algorithm description presented in Section 3.4.

Setup

Since each node in the output layer of CFRNN represented an item, we observe a significant gap between the number of output units for MovieLens 1M and the other datasets. With a total number of items of 3 706, the output layer for MovieLens 1M had considerably fewer items to choose from than with the other datasets with both total number of items above 85 000. As there can only be one correct item at each time step per user, this relatively large number of items to choose from made the sequence prediction task more complex for larger number of items. Another effect of the number of items was the number of hidden neurons selected by the grid search. We observe in case of longer sequences and more items (Am-like-ML) that 50 LSTM units in the hidden layer were preferred over 20. Note that there was no substantial difference in the performance of the grid search for Amazon 20K Users, meaning we cannot draw the same conclusion for this dataset.

The values of the diversity bias (δ) explored during the grid search were 0.01 and 0.2. The reason for not exploring larger values of δ was the idea that popular items contribute more to higher recall@n and NDCG@n scores than less popular items. In practice, however, popular items are often trivial recommendations that are either already known to users or too general to be considered as personalisation of the available items. Thus, tweaking δ could prove beneficial in practical applications of this algorithm.

Training Loss and Metrics

First of all, the average loss of CFRNN for Amazon 20K Users shows a large standard deviation and little decrease over the number of epochs. We believe that this loss, together with the difference in sequence length medians between datasets suggests the algorithm cannot learn enough from the relatively short sequences of Amazon 20K Users. More specifically, MovieLens 1M and Am-like-ML with medians of 96 and 25 respectively clearly dominate Amazon 20K Users at a median of 7. We believe this difference had a significant impact on the performance of CFRNN, even though the maximum sequence length for the dominating datasets was set to 30. Besides, the parameters selected for Amazon 20K Users were the best-performing ones on the validation set during the grid search. This implies that for any of the hyperparameter combinations in the grid search, none enabled CFRNN to actually learn to recommend.

Furthermore, we observe a monotonically decreasing loss function for MovieLens 1M, together with monotonically increasing validation metrics. With little sign of convergence in these three graphical representations, we assume the algorithm is still learning and improving in terms of performance when reaching the final epoch. In this case more epochs could result in better overall performance on MovieLens 1M.

An important difference between the previously described results and those of the Am-like-ML dataset is the difference in validation metric trends compared to the loss. For Am-like-ML we observe a monotonically decreasing loss function while the validation metrics show a relatively large standard deviation and no upward trend. We believe this is also the reason no larger number of epochs came out on top of the grid search results (Table B.11). With no signs of an upward trend in these validation metrics, we belief the algorithm quickly reached convergence in the initial epochs.

6.3 Neural Matrix Factorisation

Now for NeuMF we evaluate the results using Figure 5.3, Table 4.12, the formal notation from Section 3.1 and the experimental setup as described in Chapter 4.

Setup

As argued by Table 3 in X. He et al. (2017), a larger number of neurons in the final layer of the MLP component could lead to better performance. However, this final layer should possess an equal dimension of latent feature vectors as the GMF component since they are concatenated in the final layer of NeuMF. Since we utilise a γ equal to eight for BPR we belief using larger γ values in NeuMF would not result in a fair comparison. Without a limitation on computational resources and time it would be beneficial to the comparison if both algorithms were introduced to larger values of γ. Consequently, we would also utilise a

larger number of neurons for the MLP component in NeuMF.

In addition, the difference in number of negatives used between the different datasets suggests that exploring more values could result in different results, now only four and eight negatives were used in the grid search. Even though X. He et al. (2017) did not observe any improvement for more than four negatives for the MovieLens dataset, this threshold can be different for other datasets, as observed in this work.

Not pre-training the GMF and MLP components for NeuMFs weight initialisation can have a number of consequences. The most straightforward one being sub-optimal performance, the others will be discussed in the Training Loss and Metrics below.

Training Loss and Metrics

We observe all loss functions monotonically decreasing and the validation metrics of both Amazon 20K Users and MovieLens 1M showing an increasing trend. This indicates the algorithm was improving during its training on two out of three datasets. The loss and validation metrics of MovieLens 1M have not shown clear signs of convergence yet, indicating performance could be enhanced by increasing the number of epochs. Even though we observe an upward trend in the validation metrics for Amazon 20K Users, the exact values of recall@n and NDCG@n are still significantly distant from the values reached on the other datasets. With a converged loss function and significantly lower validation values, we suspect the algorithm learned to recommend relevant items for a small subset of the validation user population. More specifically, we belief this subset is made up of users with a relatively large number of positive user-item interactions. This because the MLP component can better express the user-item interaction mechanics when there is more data available per user.

For Am-like-ML we observe a highly volatile start in terms of recall@n and NDCG@n, quickly reaching the observed maximum for both metrics before decreasing for the rest of the epochs (Figure 5.3b, 5.3c). We believe this behaviour can be explained by the large number of negatives with which the algorithm was trained. The large standard deviation at the beginning of training can be due to not initialising NeuMF with pre-trained weights of both its components. This means initialisation happens according to the initialisation methods as described in Table 4.11 which, as pointed out by X. He et al. (2017), can have a significant impact on the results. With eight negatives per positive item, the algorithm gets to observe many examples to learn from and therefore quickly reached peak performance around epoch seven for the validation metrics. The downward trend after the peak in both validation metrics is most likely the downside of having a large number of negatives in the training samples as the model starts to overfit.

Note that each sample for this algorithm contains all positive interactions, each

matched with a number of negative interactions per user. Since the negative items are randomly sampled the performance of this algorithm may differ for different samples.

6.4 Performance Comparison

With clear insights into each model's setup and training procedure, we now focus on the comparison of the held-out test set results per dataset, as shown in Figure 5.4. This section follows the structure of the research questions, meaning we compare the deep learning based models first. Next, each of these models is compared with the MF benchmark for all datasets. Recall that in this work, an algorithm only outperforms another algorithm when the average recall@n and average NDCG@n are proven to be statistically different from one another (Table 5.1, 5.2) and both scores of one algorithm are above those of the other, for all n (Figure 5.4).

6.4.1 CFRNN vs. NeuMF

As explained before, we believe the combination of insufficient sequence lengths together with a substantial total number of items led to CFRNN's poor performance on the Amazon 20K Users dataset. For NeuMF we did observe monotonically increasing validation metrics and a clear decrease in loss. We believe the MLP component becomes more of a liability than an advantage when there is not enough data available. The GMF part, which is based on standard MF, can still learn in this setting as shown by BPR. The concatenation within the final layer of NeuMF combines its components, lowering GMF's performance with the MLP component. This explains the small, yet significant gain of NeuMF over CFRNN in terms of both recall@n (Table 5.1) and NDCG@n (Table 5.2).

On MovieLens 1M we observe similar performance in terms of both recall@n and NDCG@n for these algorithms. These findings are in line with both Devooght and Bersini (2016) and X. He et al. (2017) for CFRNN and NeuMF, respectively. These authors utilised the same MovieLens 1M dataset for introducing their respective algorithms and showcased the improvement over standard methods like K-Nearest Neighbour and BPR. Even though our training, test and validation sets differ from the aforementioned research in terms of structure, we still observe how these deep learning algorithms thrive under the right circumstances. Thus, in terms of both metrics we cannot select one algorithm that clearly dominates the other graphically, let alone in terms of statistical significant differences between their results (Table 5.1, 5.2).

Finally, the results of the combination of both datasets reveals both the shortcoming of sequential interpretation of the recommender problem used by CFRNN, and the advantage of combining deep learning with MF used by NeuMF. With most sequences below the maximum sequence length in Am-like-ML, we still

observe adequate performance of CFRNN. NeuMF, however, clearly dominates CFRNN as it can easily build the user latent feature vectors with the available user-item interactions per user.

This partly answers **SQ3** as the different structures of the data clearly influence the results of these deep learning algorithms. For CFRNN we state that with shorter sequences and more items, the results experience a setback in performance compared to its results on a dataset like MovieLens 1M. Its performance degrades when user-item interaction sequences are shortened and the sequences are made up of many different items. NeuMF can clearly utilise the vast amount of user-item interactions for MovieLens 1M and Am-like-ML to build the user latent feature vectors. It has become apparent that with fewer interactions, as in Amazon 20K Users, this algorithm still outperforms the other deep learning approach. We believe this outperformance is due to the GMF component being able to learn from the relatively short sequences while being hindered by the MLP part of the algorithm. Therefore, in terms of absolute recall@n and NDCG@n scores NeuMF's performance is still distant from the non-deep learning algorithm.

6.4.2 BPR vs. CFRNN

Based on the same intuition as before, we belief CFRNN is not able to learn from the relatively short sequences of users in Amazon 20K Users. BPR on the other hand shows it is still able to learn, even with little user-item interactions per user. While this learning might be slower compared to MovieLens 1M and Am-like-ML, as mentioned in Section 6.1, we observe dominance of BPR over CFRNN in terms of recall@n and NDCG@n for every n (Table 5.1, Table 5.2). We belief the difference in optimisation criterion between CFRNN and BPR is what creates this substantial gap in performance for the Amazon 20K Users dataset. Where CFRNN has to pick the one correct item, out of many items, at every time step, BPR updates the user latent feature vectors based on a comparison between two items. This means that during training, CFRNN sees all non-interacted items as negative instances (0), while the positive instances are represented as a one on each time step. The pairwise optimisation of BPR assumes the positive items are preferred over the negative items, but does not imply the negative item is actually disliked. This difference is crucial for the performance of the algorithms when there are little interactions available as can be seen in Figure 5.4a.

With more user-item interactions per user, as in MovieLens 1M, we observe vastly different performance for CFRNN. As mentioned before, the length of the sequences has a substantial impact on CFRNN's performance. We believe the aforementioned negative representation of non-interacted items is mitigated by the fact that each item occurs more frequently, meaning each item is more frequently observed as a one during training for MovieLens 1M than in the case of Amazon 20K Users. This results in CFRNN outperforming BPR both graph-

ically as shown in Figure 5.4b and statistically significant as can be observed in Tables 5.1 and 5.2. BPR still shows adequate performance, but could have better represented the user and item latent feature vectors now that there is more data available per user and item. In other words, we belief BPR's performance can be enhanced with a larger value of γ in the case of MovieLens 1M.

With the combination of the aforementioned datasets in Am-like-ML, we also believe a combination of the aforementioned algorithm characteristics affect the results (Figure 5.4c). Even though CFRNN trained on relatively long user-item interaction sequences, the frequency of items being observed as a zero during training is increased compared to MovieLens 1M. This is due to the larger number of items and the far lower number of interactions per item in Am-like-ML. Thus, while still showing adequate performance in Figure 5.4c we observe CFRNN being outperformed by BPR in both recall@n and NDCG@n as shown in 5.1 and 5.2.

Based on the previously explained insights we continue on **SQ3**. In this comparison, the difference in objective functions together with the structural differences in the datasets play a key part in the difference in performance of the algorithms. Taking this into account, we believe the pairwise ranking optimisation criterion of BPR shows more robust performance on datasets with different structures. While for CFRNN, the negative representation of non-interacted items seems to fade when the items are frequently observed as a one during training. Furthermore, the sequence length is one factor in CFRNN's performance, while the combination of high item interaction frequency and long sequences shows it can outperform the MF benchmark of this research.

6.4.3 BPR vs. NeuMF

As already mentioned, we belief NeuMF needs more user-item interactions than present in Amazon 20K Users to perform as it does on the other datasets. Note that NeuMF is a combination of a generalised form of MF and an MLP that update user and item latent feature factors. As observed in the performance on the other datasets, this MLP component provides NeuMF with an edge over BPR when there is enough data available per user. Again, BPR's pairwise ranking objective function largely contributes to its success on the Amazon 20K Users dataset. With this being said, we observe that BPR clearly outperforms NeuMF on Amazon 20K Users as shown in Figure 5.4a, Table 5.1 and Table 5.2.

On MovieLens 1M, we observe similar domination of NeuMF over BPR as with CFRNN, in terms of recall@n and NDCG@n (Figure 5.4b). This is in line with the findings presented in X. He et al. (2017), where NeuMF also outperforms BPR. Even though the pairwise ranking optimisation makes BPR relatively robust to the underlying dataset, NeuMF performs better depending on the underlying structure of the data.

This is also observed for the Am-like-ML dataset in Figure 5.4c, where NeuMF has the upper hand when it comes to ranking (NDCG@n). Looking at the p-values in Table 5.1 we cannot declare a clear winner in terms of recall@n. Again NeuMF benefited from its ability to represent the user-item interactions in a non-linear way with its MLP component. From the correctly classified items (within top 20 recommendations), many of these items are ranked at rank@1 compared to BPR. This indicates well-represented users and items in their latent feature spaces of NeuMF as there are many items to choose from. Note that NeuMF might also have benefited from a larger number of negatives in this particular case.

Finally we incorporate this information in the answer to **SQ3**. Both this work and the aforementioned research show that combining a generalised form of MF with an MLP component can lead to enhanced performance compared to standard MF optimised using BPR. However, as shown in this work, with less interactions per user and more items we observe a decrease in NeuMF's performance compared to BPR. We believe the MLP component of NeuMF can restrain the performance of GMF, as the final layer of NeuMF consists of a concatenation of both components. This deduction seems to be reinforced by the performance of NeuMF on Amazon 20K Users, where it still outperforms CFRNN but does not come close to BPR for both evaluation metrics.

Chapter 7

Conclusions and Future Work

In this final chapter, we reiterate the research questions and provide concise answers, based on the findings of this work. Secondly, we summarise the reasons for the differences in performance based on the data structures and algorithm architectures. In addition, we convert the technical reasons for these differences to managerial implications to guide decision making within *YGroup*. Finally, we consider topics and open questions that future research can address.

7.1 Research Questions

In this thesis, we compared *Bayesian Personalised Ranking* (BPR), *Collaborative Filtering with Recurrent Neural Networks* (CFRNN) and *Neural Matrix Factorisation* (NeuMF) in terms of recall@n and NDCG@n on the well-known MovieLens 1M, our Amazon 20K Users and our Am-like-ML datasets. Am-like-ML is a combination of the other datasets that provides us with more insights regarding model performance. Even though these datasets are originally rating datasets, they have been used as implicit feedback data to align with *YGroup*'s objectives. The algorithms have been optimised using a grid search and trained in a similar fashion as in Devooght and Bersini (2016). The final comparison is based on the average recall@n and NDCG@n of 30 runs per algorithm for $n \in \{1, 5, 10, 15, 20\}$. The research questions as posed in the beginning of this thesis are reiterated below.

RQ1 How do *Collaborative Filtering with Recurrent Neural Networks* and *Neural Network based Collaborative Filtering* compare to each other in terms of recommendation performance on fashion and movie datasets?

RQ2 How do these deep learning models perform compared to a *Matrix Factorisation* benchmark model in terms of recommendation performance on fashion and movie datasets?

In the comparison of CFRNN and NeuMF, both showed similar performance on MovieLens 1M; however, NeuMF outperforms CFRNN on both Amazon 20K Users and Am-like-ML. Thus to answer **RQ1**, NeuMF surpassed CFRNN in both recorded metrics for the fashion datasets, whereas both algorithms performed equally well on the MovieLens 1M dataset.

For **RQ2**, we observed outperformance of BPR by both deep learning algorithms on the MovieLens 1M data, which is in inline with both Devooght and Bersini (2016) (CFRNN) and X. He et al. (2017) (NeuMF). For Amazon 20K Users we observed the opposite results, both deep learning algorithms were surpassed in terms of recall@n and NDCG@n by BPR. For the structural mix of MovieLens 1M and Amazon 20K Users; Am-like-ML, we observed a mix of the previously mentioned results. NeuMF dominates the other algorithms in terms of ranking and shows significantly higher rank@1 scores for both metrics.

7.2 Conclusions

One of our main contributions is to exhibit the difference in recommendation performance of these models on the differently structured datasets. In addition, we showcase the impact of structural differences between datasets from both fashion and movies on the performance of the aforementioned models.

While evaluation metrics can widely differ between researchers as mentioned in Chapter 2, this work is the first to compare these algorithms in terms of a classification and a ranking metric based on all items. This comparison has shown similar performance behaviour on the well-known MovieLens 1M dataset with *Matrix Factorisation* (MF) based BPR as presented in Devooght and Bersini (2016) (CFRNN) and X. He et al. (2017) (NeuMF). The novel datasets with a different underlying structure revealed a shift in the performance of both deep learning based algorithms to be either outperformed by or comparable with BPR. The visual representation of these results is shown in Figure 5.4 while a detailed description of the underlying reasons is provided in Section 6.4.

This work shows BPR's pairwise ranking criterion can be advantageous compared to the multi-class classification optimisation of CFRNN when there are little user-item interactions and many items in total. In addition, while the combination of NeuMF's linear GMF and its non-linear MLP components prove beneficial for datasets with well-represented users and items in terms of interactions. We have reason to believe the MLP part can be a drawback in scenarios where users possess less interactions with items.

With the observed performance of each model on the different datasets, we can recommend each algorithm in a different setting. First of all, BPR can be utilised in a broad range of CF recommendation problems, as a baseline form of recommending. Its pairwise ranking optimisation shows robust performance

for the three different datasets explored in this research. Secondly, when there is a substantial amount of purchase history data available for a large number of users, one could argue for CFRNN as it shows promising performance on MovieLens 1M. Note that these types of datasets can be specific to a certain domain, such as popular streaming platforms. Finally, if there is only a subset of users with a considerable amount of purchase history data, we believe NeuMF can be the optimal choice. In practice this can be utilised on a subset of users who represent most of the user-item interactions, such as premium users or frequent buyers. Since NeuMF updates individual user representations, it is able to better represent the user-item interactions for longer user-item interaction sequences, as observed in its Am-like-ML results.

As optimisation is not the goal of this work we suggest optimising each algorithm to the problem at hand before selecting an approach.

7.3 Future Work

While the experimental setup and the corresponding results in this thesis yield novel insights that build-up on previous findings, further investigations could lead to valuable refinements. In general, a broader grid search could also broaden the insights in model performance on the proposed datasets. In addition, enlarging the training and validation sets or incorporating a form of cross-validation could provide a more generalised view of the results compared to the ones obtained in this thesis. In terms of data, it could be of added value to include a dataset belonging to a different industry with different structural characteristics to have an additional measure of performance for each algorithm.

In more detail, training and testing the sample-based algorithms (BPR and NeuMF) on multiple sets of different samples could produce more insights in the impact of sampling on these algorithms. More specifically, we belief BPR's performance can differ when trained and tested on a different set of samples. Even though X. He et al. (2017) utilised binary cross-entropy loss to optimise NeuMF, they argue many different optimisation criteria can be adopted. To this end, we belief NeuMF would benefit from adopting BPR's pairwise ranking criterion for optimisation. In addition, incorporating a hyperparameter to specifically weight the contribution of each of the two components within NeuMF could prove to be beneficial when the target data is structured as Amazon 20K Users. Finally, to verify our deduction that the MLP component could restrain GMF in certain situations, one can train and test the individual components of NeuMF and compare their results.

As shown in Figure 4 of Devooght and Bersini (2016) one can use GRU units instead of LSTM units or utilise a bidirectional LSTM or 2-layered LSTM structure. Even though the difference in model structure did not reveal significantly different results in the aforementioned research, they could prove to be beneficial

for the metrics or the datasets used in this research. Another difference between Devooght and Bersini (2016) in terms of performance comparison and this work is the evaluation metrics used. They propose a novel metric, named sps@10, which measures the short-term prediction success of their CFRNN algorithm. Expressing our results in terms of their sps@10 could provide additional insights. Furthermore, X. He et al. (2017) compare their NeuMF with BPR using an item sampling approach. Their evaluation metric places a positive item for user u in a sample of negative items for that user. Then they compare existing algorithms with their novel GMF, MLP and NeuMF in terms of hitrate@10 and NDCG@10. Using this item sampling-based approach for performance evaluation, we could observe different behaviour for both BPR and NeuMF (infeasible for CFRNN). Finally, with the success of NeuMF on two out of three datasets, we would like to investigate a similarly structured algorithm dubbed ConvNCF (X. He et al., 2018).

Bibliography

Amazon review data. (2018). https://nijianmo.github.io/amazon/index.html. (Accessed: 2020-04-26)

Bell, R. M., Koren, Y., & Volinsky, C. (2010). All together now: A perspective on the netflix prize. *Chance*, *23*(1), 24–29.

Bennett, J., & Lanning, S. (2007). The netflix prize. In *Proceedings of kdd cup and workshop* (Vol. 2007, p. 35).

Bhulai, S. (2018). *Advanced machine learning lecture 7: Recurrent neural networks.* https://canvas.vu.nl/courses/36744/files?preview=887860. (Lecture slides: 2018-09-28)

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, *12*(4), 331–370.

Chen, G. (2016). A gentle tutorial of recurrent neural network with error backpropagation. *CoRR*, *abs/1610.02583*. Retrieved from http://arxiv.org/abs/1610.02583

Chen, H. (2017). Weighted-svd: Matrix factorization with weights on the latent factors. *CoRR*, *abs/1710.00482*. Retrieved from http://arxiv.org/abs/1710.00482

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., ... Ispir, M. (2016). Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems* (p. 7–10). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2988450.2988454 doi: 10.1145/2988450.2988454

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, *abs/1409.1259*. Retrieved from http://arxiv.org/abs/1409.1259

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, *abs/1406.1078*. Retrieved from http://arxiv.org/abs/1406.1078

Chung, J., Gülçehre, Ç., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, *abs/1412.3555*. Retrieved from http://arxiv.org/abs/1412.3555

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., ... et al. (2010). The youtube video recommendation system. In *Proceedings of the fourth acm conference on recommender systems* (p. 293–296). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1864708.1864770 doi: 10.1145/1864708.1864770

Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, *3*, 1–29.

Devooght, R., & Bersini, H. (2016). Collaborative filtering with recurrent neural networks. *CoRR*, *abs/1608.07400*. Retrieved from http://arxiv.org/abs/1608.07400

Donkers, T., Loepp, B., & Ziegler, J. (2017). Sequential user-based recurrent neural network recommendations. In *Proceedings of the eleventh acm conference on recommender systems* (p. 152–160). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3109859.3109877 doi: 10.1145/3109859.3109877

Duchi, J., Hazan, E., & Singer, Y. (2011, July). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, *12*(null), 2121–2159.

Dziugaite, G. K., & Roy, D. M. (2015). Neural network matrix factorization. *CoRR*, *abs/1511.06443*. Retrieved from http://arxiv.org/abs/1511.06443

eMarketer. (2017). *Worldwide retail and ecommerce sales: emarketer's estimates for 2016–2021.* https://www.emarketer.com/Report/Worldwide-Retail-Ecommerce-Sales-eMarketers-Estimates-20162021/2002090. (Accessed:2020-03-21)

Glorot, X., & Bengio, Y. (2010, 13–15 May). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR. Retrieved from http://proceedings.mlr.press/v9/glorot10a.html

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In G. J. Gordon, D. B. Dunson, & M. Dudík (Eds.), *Aistats* (Vol. 15, p. 315-323). JMLR.org. Retrieved from http://dblp.uni-trier.de/db/journals/jmlr/jmlrp15.html#GlorotBB11

Gomez-Uribe, C. A., & Hunt, N. (2016, December). The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, *6*(4), 13:1-13:19. Retrieved from https://doi.org/10.1145/2843948 doi: 10.1145/2843948

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT press.

Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). Deepfm: A factorization-machine based neural network for CTR prediction. *CoRR*, *abs/1703.04247*. Retrieved from http://arxiv.org/abs/1703.04247

Hallinan, B., & Striphas, T. (2016). Recommended for you: The netflix prize and the production of algorithmic culture. *New media & society*, *18*(1), 117–137.

He, R., & McAuley, J. (2016a). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th international conference on world wide web* (p. 507–517). Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee. Retrieved from https://doi.org/10.1145/2872427.2883037 doi: 10.1145/2872427.2883037

He, R., & McAuley, J. (2016b). Vbpr: Visual bayesian personalized ranking from implicit feedback. In *Proceedings of the thirtieth aaai conference on artificial intelligence* (p. 144–150). AAAI Press.

He, X., & Chua, T.-S. (2017). Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th international acm sigir conference on research and development in information retrieval* (p. 355–364). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3077136.3080777 doi: 10.1145/3077136.3080777

He, X., Du, X., Wang, X., Tian, F., Tang, J., & Chua, T. (2018). Outer product-based neural collaborative filtering. *CoRR*, *abs/1808.03912*. Retrieved from http://arxiv.org/abs/1808.03912

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (p. 173–182). Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee. Retrieved from https://doi.org/10.1145/3038912.3052569 doi: 10.1145/3038912.3052569

Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

Hidasi, B., Quadrana, M., Karatzoglou, A., & Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th acm conference on recommender systems* (p. 241–248). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2959100.2959167 doi: 10.1145/2959100.2959167

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107–116.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 eighth ieee international conference on data mining* (pp. 263–272).

Jing, H., & Smola, A. J. (2017). Neural survival recommender. In *Proceedings of the tenth acm international conference on web search and data mining*

(p. 515–524). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3018661.3018719 doi: 10.1145/3018661.3018719

Kang, H., & Yoo, S. J. (2007). Svm and collaborative filtering-based prediction of user preference for digital fashion recommendation systems. *IEICE transactions on information and systems*, *90*(12), 2100–2103.

Kim, D., Park, C., Oh, J., Lee, S., & Yu, H. (2016). Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th acm conference on recommender systems* (pp. 233–240). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2959100.2959165 doi: 10.1145/2959100.2959165

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings.* Retrieved from http://arxiv.org/abs/1412.6980

Klema, V., & Laub, A. (1980). The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, *25*(2), 164–176.

Kombrink, S., Mikolov, T., Karafiát, M., & Burget, L. (2011, 01). Recurrent neural network based language modeling in meeting recognition. In (Vol. 11, p. 2877-2880).

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *Proceedings of the 15th acm sigkdd international conference on knowledge discovery and data mining* (p. 447–456). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1557019.1557072 doi: 10.1145/1557019.1557072

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

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, *37*(2), 233-243. Retrieved from https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209 doi: 10.1002/aic.690370209

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient backprop. In G. B. Orr & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (pp. 9–50). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/3-540-49430-8_2 doi: 10.1007/3-540-49430-8_2

Lei, C., Liu, D., Li, W., Zha, Z., & Li, H. (2016). Comparative deep learning of hybrid representations for image recommendations. *CoRR*, *abs/1604.01252*. Retrieved from http://arxiv.org/abs/1604.01252

Li, Z., Zhao, H., Liu, Q., Huang, Z., Mei, T., & Chen, E. (2018). Learning from history and present: Next-item recommendation via discriminatively exploiting user behaviors. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery data mining* (p. 1734–1743). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3219819.3220014

doi: 10.1145/3219819.3220014

Liao, K. (2019). *Prototyping a recommender system step by step part 2: Alternating least square (als) matrix factorization in collaborative filtering.* https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1. (accessed: 2020-07-13)

Movielens 1m data. (2003). https://grouplens.org/datasets/movielens/1m/. (Accessed: 2020-07-13)

Movielens 25m data. (2019). https://grouplens.org/datasets/movielens/25m/. (Accessed: 2020-06-25)

Ncf framework. (2018). https://github.com/hexiangnan/neural_collaborative_filtering. (Accessed: 2020-07-16)

Netflix 100m data. (2019). https://www.kaggle.com/netflix-inc/netflix-prize-data. (accessed: 2020-04-26)

Ni, J., Li, J., & McAuley, J. (2019, November). Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 188–197). Hong Kong, China: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/D19-1018 doi: 10.18653/v1/D19-1018

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, *abs/1811.03378*. Retrieved from http://arxiv.org/abs/1811.03378

Olah, C. (2015). *Understanding lstm networks.* http://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed: 2020-05-19)

Rendle, S. (2010). Factorization machines. In *Proceedings of the 2010 ieee international conference on data mining* (p. 995–1000). USA: IEEE Computer Society. Retrieved from https://doi.org/10.1109/ICDM.2010.127 doi: 10.1109/ICDM.2010.127

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2012). BPR: bayesian personalized ranking from implicit feedback. *CoRR*, *abs/1205.2618*. Retrieved from http://arxiv.org/abs/1205.2618

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton project para.* Cornell Aeronautical Laboratory.

Samet, A. (2020). *Us ecommerce will rise 18% in 2020 amid the pandemic.* https://www.emarketer.com/content/us-ecommerce-will-rise-18-2020-amid-pandemic?ecid=NL1001/. (Accessed: 2020-07-13)

Sedhain, S., Menon, A. K., Sanner, S., & Xie, L. (2015). Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on world wide web* (p. 111–112). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2740908.2742726 doi: 10.1145/2740908.2742726

Shepherd, A. J. (2012). Second-order methods for neural networks: Fast and re-

liable training methods for multi-layer perceptrons. In (p. 16-17). Springer Science & Business Media.

Smirnova, E., & Vasile, F. (2017). Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd workshop on deep learning for recommender systems* (p. 2–9). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3125486.3125488` doi: 10.1145/3125486.3125488

Smith, B., & Linden, G. (2017). Two decades of recommender systems at amazon. com. *Ieee internet computing*, *21*(3), 12–18.

Strub, F., Gaudel, R., & Mary, J. (2016). Hybrid recommender system based on autoencoders. In *Proceedings of the 1st workshop on deep learning for recommender systems* (p. 11–16). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2988450.2988456` doi: 10.1145/2988450.2988456

Sutskever, I., Martens, J., & Hinton, G. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on international conference on machine learning* (p. 1017–1024). Madison, WI, USA: Omnipress.

Wang, H., Wang, N., & Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems. In *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining* (p. 1235–1244). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2783258.2783273` doi: 10.1145/2783258.2783273

Wu, C.-Y., Ahmed, A., Beutel, A., Smola, A. J., & Jing, H. (2017). Recurrent recommender networks. In *Proceedings of the tenth acm international conference on web search and data mining* (p. 495–503). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3018661.3018689` doi: 10.1145/3018661.3018689

Wu, S., Ren, W., Yu, C., Chen, G., Zhang, D., & Zhu, J. (2016). Personal recommendation using deep recurrent neural networks in netease. In *2016 ieee 32nd international conference on data engineering (icde)* (pp. 1218–1229). doi: 10.1109/ICDE.2016.7498326

Yu, W., Zhang, H., He, X., Chen, X., Xiong, L., & Qin, Z. (2018). Aesthetic-based clothing recommendation. In *Proceedings of the 2018 world wide web conference* (p. 649–658). Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee. Retrieved from `https://doi.org/10.1145/3178876.3186146` doi: 10.1145/3178876.3186146

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, *52*(1), 1–38.

Zheng, L., Noroozi, V., & Yu, P. S. (2017). Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the tenth acm international conference on web search and data mining* (p. 425–434). New York, NY, USA: Association for Computing Ma-

chinery. Retrieved from https://doi.org/10.1145/3018661.3018665 doi: 10.1145/3018661.3018665

Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. (2008). Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th international conference on algorithmic aspects in information and management* (p. 337–348). Berlin, Heidelberg: Springer-Verlag. Retrieved from https://doi.org/10.1007/978-3-540-68880-8_32 doi: 10.1007/978-3-540-68880-8_32

Appendix A

Data Specifications

A.1 Full Data Characteristics

A.1.1 MovieLens 25M

The dataset can be found on *MovieLens 25M data* (2019), below are its *Exploratory Data Analysis* (EDA), features used and data specifications in Figure A.1, Table A.1 and Table A.2, respectively.

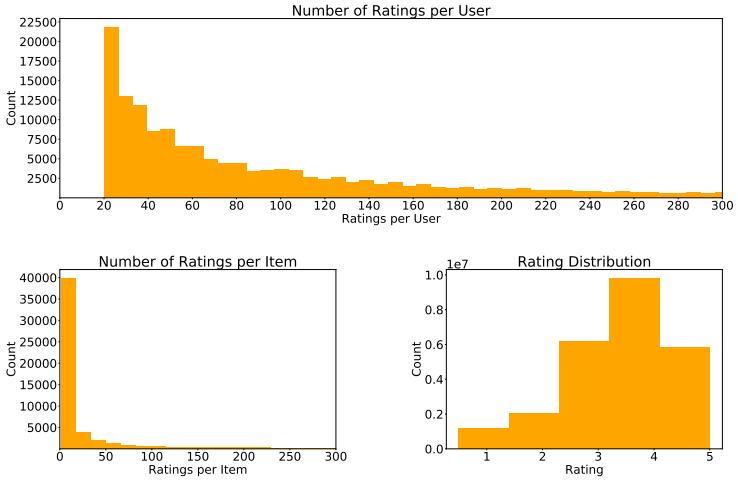


Figure A.1: Exploratory Data Analysis for MovieLens 25M dataset

Table A.1: MovieLens 25M features used

Original Name	Used Name
rating	rating
user	user_id
item	item_id
datetime	datetime

Table A.2: MovieLens 25M specifics

General Statistics	Value
Total Interactions	25 000 000
Total Users	162 541
Total Items	59 047
Sparseness	99.998%
Average Rating	3.53/5
Interactions Per User	
Average	153.81
Median	71.0
Interactions Per Item	
Average	423.39
Median	6.0

A.1.2 Amazon 5-core Clothing Shoes and Jewellery

The dataset can be found on *Amazon Review data* (2018), below are its *Exploratory Data Analysis* (EDA), features used and data specifications in Figure A.2, Table A.3 and Table A.4, respectively.

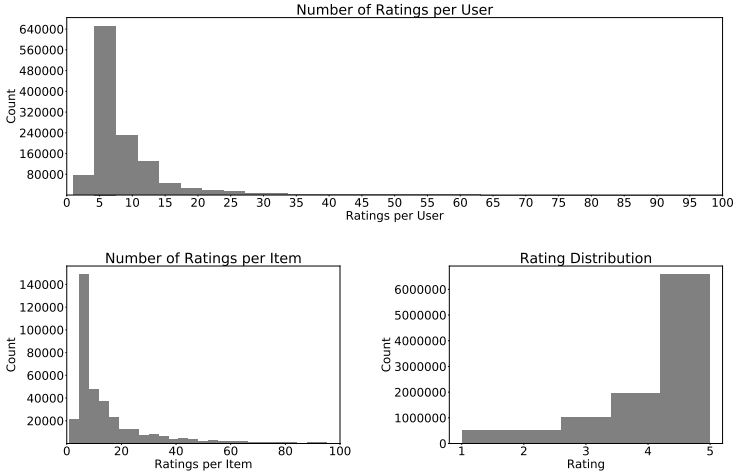


Figure A.2: Exploratory Data Analysis for Amazon Shoes Clothing and Jewellery dataset

Table A.3: Amazon Clothing Shoes and Jewellery features used

Original Name	Used Name
overall	rating
vote	-
verified	-
reviewTime	-
reviewerID	user_id
asin	item_id
style	-
reviewerName	-
reviewText	-
summary	-
unixReviewTime	datetime
image	-

Table A.4: Amazon Clothing Shoes and Jewellery specifics

General Statistics	Value
Total Interactions	11 285 464
Total Users	1 219 678
Total Items	376 858
Sparseness	99.998%
Average Rating	4.28/5
Interactions Per User	
Average	9.25
Median	7.0
Interactions Per Item	
Average	29.95
Median	10.0

A.1.3 Comparison

Here we put the characteristics of the full datasets next to each other.

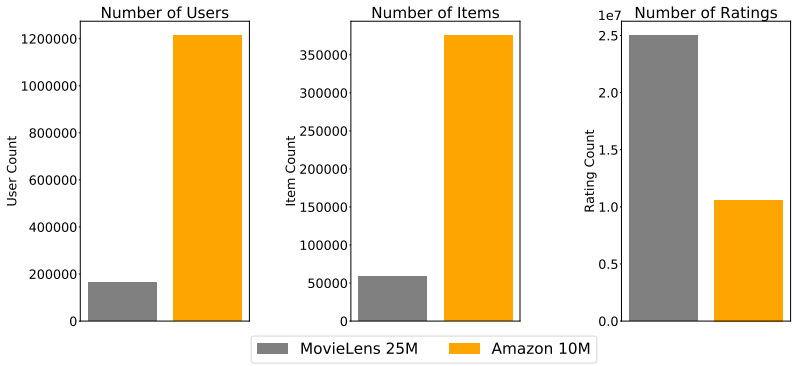


Figure A.3: Comparison of the number of users, items and ratings for Amazon Shoes Clothing and Jewellery dataset and the MovieLens 25M dataset

A.2 Ratings per User & Item

Here we provide a more detailed view of the long tail of the number of ratings and number of items per dataset.

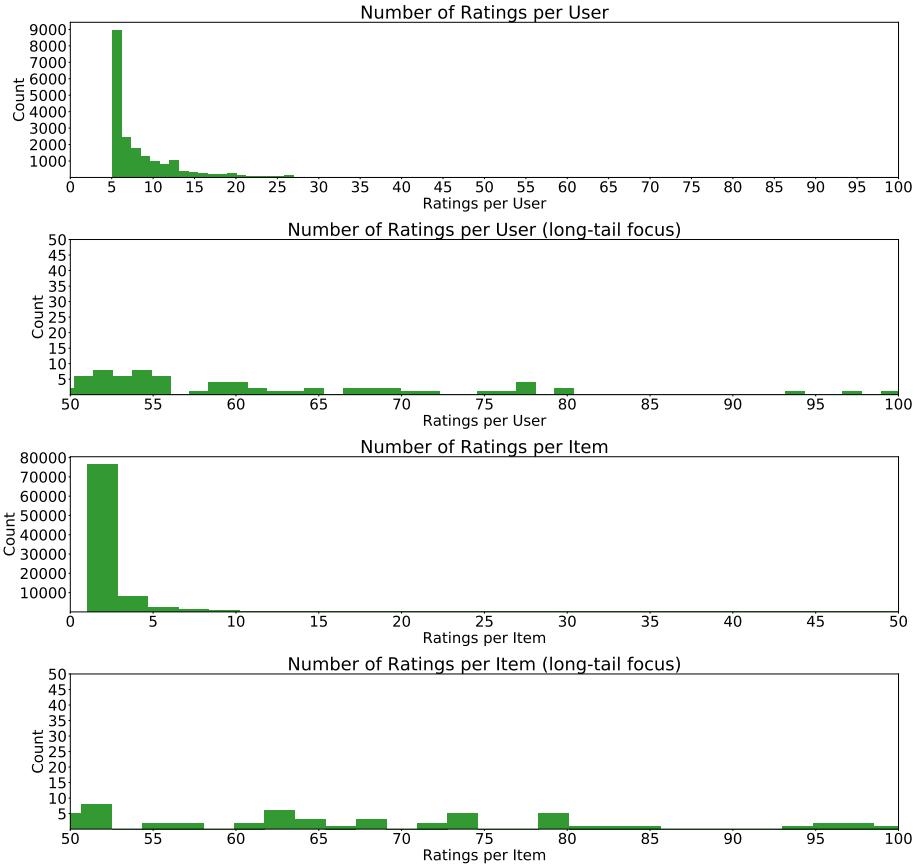


Figure A.4: Number of ratings per user and number of ratings per item, with their long-tail focused representation for Amazon 20K Users

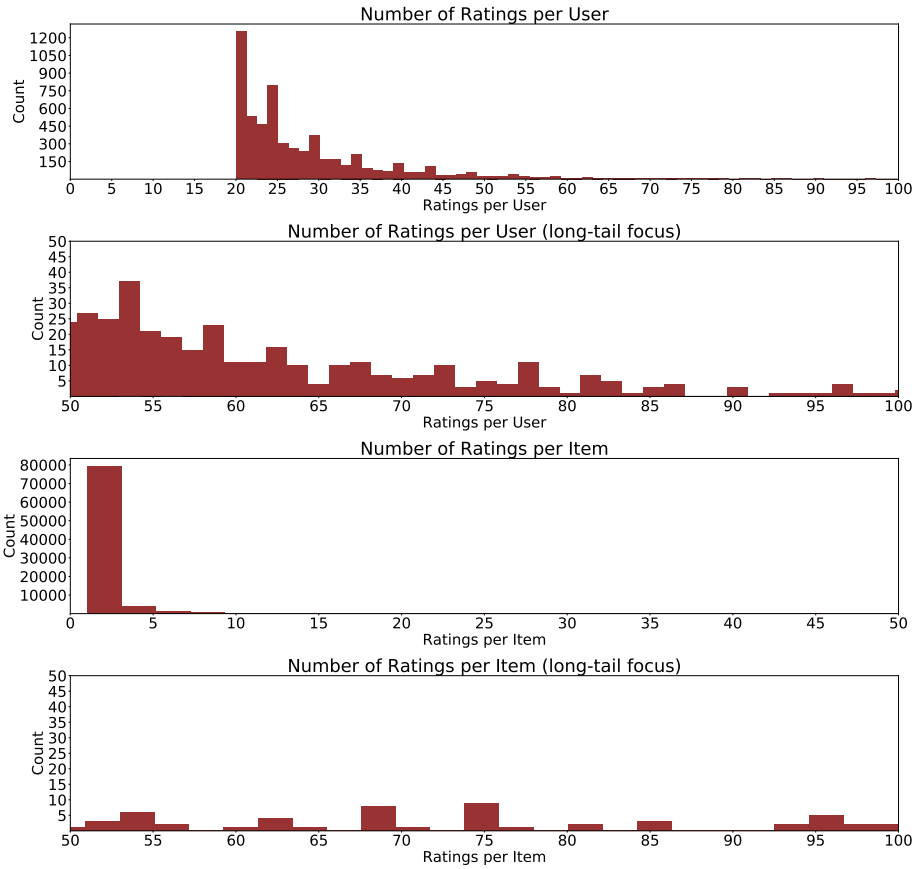


Figure A.5: Number of ratings per user and number of ratings per item, with their long-tail focused representation for Am-like-ML

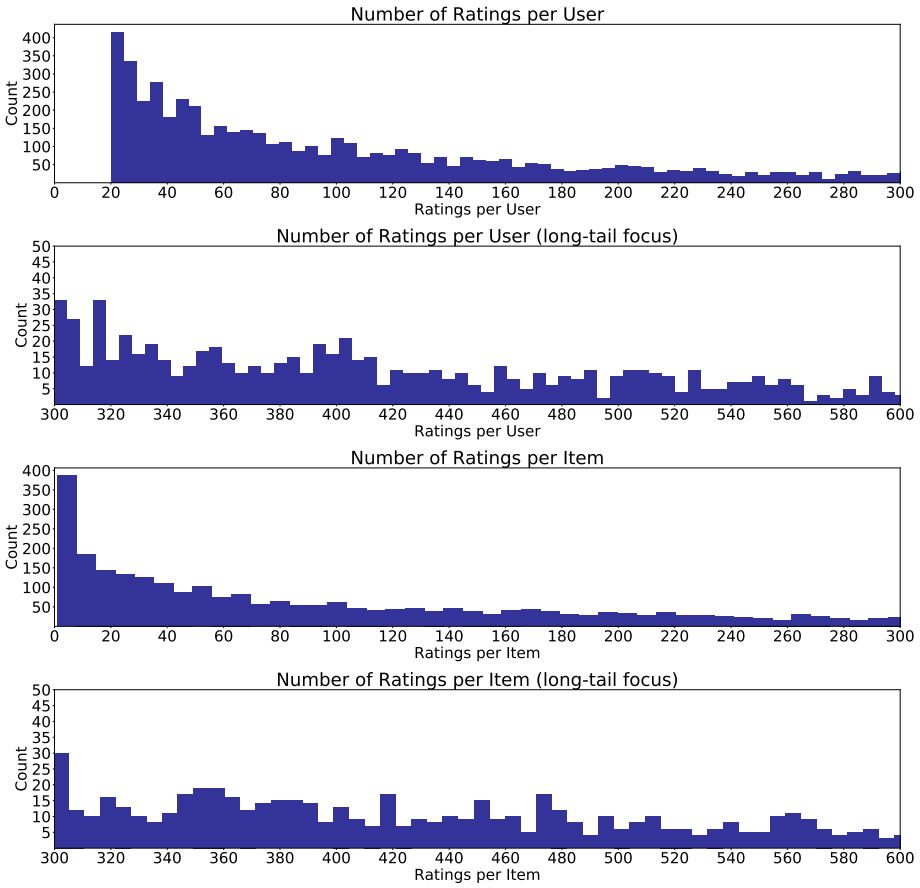


Figure A.6: Number of ratings per user and number of ratings per item, with their long-tail focused representation for MovieLens 1M

Appendix B

Grid Search

B.1 Parameters

All parameters explored during the grid search for each algorithm and every dataset.

Table B.1: BPR parameters used for the grid search per dataset

Parameters	Amazon 20k Users	MovieLens 1M	Am-Like-ML
γ	8	8	8
Epochs	25	25	25
α	0.01, 0.03, 0.05, 0.08, 0.1, 0.12, 0.15	0.01, 0.03, 0.05, 0.08, 0.1, 0.12, 0.15	0.01, 0.03, 0.05, 0.08, 0.1, 0.12, 0.15
ρ	1.05	1.05	1.05
σ	0.55	0.55	0.55
λ_p	0, 0.001, 0.01, 0.1, 0.2	0, 0.001, 0.01, 0.1, 0.2	0, 0.001, 0.01, 0.1, 0.2
λ_q	0, 0.001, 0.01, 0.1, 0.2	0, 0.001, 0.01, 0.1, 0.2	0, 0.001, 0.01, 0.1, 0.2
Sample% of interactions	10%, 30%, 50%, 80%	10%, 30%, 50%, 80%	10%, 30%, 50%, 80%

Table B.2: NeuMF parameters used for the grid search per dataset

Parameters	Amazon 20k Users	MovieLens 1M	Am-Like-ML
γ	8	8	8
Layers	16, 32, 16, 8	16, 32, 16, 8	16, 32, 16, 8
Epochs	20	20	20
α	0.00001, 0.00005, 0.0001, 0.0005	0.00001, 0.00005, 0.0001, 0.0005	0.00001, 0.00005, 0.0001, 0.0005
Batch Size	256, 512	256, 512	256, 512
#Negatives	4, 8	4	4, 8

Table B.3: CFRNN parameters used for the grid search per dataset

Parameters	Amazon 20k Users	MovieLens 1M	Am-Like-ML
δ	0.01, 0.2	0.01, 0.2	0.01, 0.2
RNN Units	20, 50	20	20, 50
Epochs	20, 50, 100	20, 50, 100	20, 50, 100
α	0.05, 0.1, 0.2	0.05, 0.1, 0.2	0.05, 0.1, 0.2
Batch Size	16, 32, 64	16, 32, 64	16, 32, 64
Max Sequence Length	10, 20, 30	10, 20, 30	10, 20, 30
Embedding Dimension	100	100	100

B.2 Grid Search Results

The following tables show the variables tracked during the Grid Search of the top 5 parameter combinations ranked on recall@10 for each algorithm on every dataset used.

Table B.4: Top 5 grid search parameter sets ranked on validation recall@10 of BPR on MovieLens 1M

train_time	total_val_rec	val_rec@10	nolf	n_iterations	sample_size	learning_rate	rho	sigma	reg_user	reg_item
157.5819	0.396	0.092	8	25	99870.9	0.05	1.1	0.5	0.001	0.001
344.8704	0.394	0.088	8	25	299612.7	0.03	1.1	0.5	0	0
332.731	0.39	0.086	8	25	299612.7	0.03	1.1	0.5	0.0001	0.0001
787.6276	0.372	0.086	8	25	798967.2	0.1	1.1	0.5	0.01	0.01
534.8713	0.404	0.086	8	25	499354.5	0.12	1.1	0.5	0.01	0.01

Table B.5: Top 5 grid search parameter sets ranked on validation recall@10 of CFRNN on MovieLens 1M

val_rec@10	total_val_rec	train_time	epochs	BATCH_SIZE	learning_rate	delta	max_seq_len	embedding_dim	rnn_units	pad_value
0.066	0.314	285.5414	100	16	0.2	0.01	30	100	20	3706
0.066	0.288	24.49307	20	64	0.05	0.01	30	100	20	3706
0.06	0.302	283.0391	100	16	0.1	0.2	30	100	20	3706
0.054	0.274	285.0395	100	16	0.05	0.01	30	100	20	3706
0.054	0.258	30.22245	20	64	0.1	0.01	30	100	20	3706

Table B.6: Top 5 grid search parameter sets ranked on validation recall@10 of NeuMF on MovieLens 1M

total_val_rec	val_rec@10	train_time	learning_rate	batch_size	layers	reg_layers	reg_mf	nolf	epochs	num_neg
0.426	0.096	621.7876	0.0001	256	[16, 32, 16, 8]	[0, 0, 0, 0]	[0, 0]	8	20	4
0.438	0.096	937.3461	0.0001	256	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[0, 0]	8	30	4
0.418	0.094	617.2445	0.0001	256	[16, 32, 16, 8]	[1e-05, 1e-05, 1e-05, 1e-05]	[0, 0]	8	20	4
0.412	0.092	614.7509	0.0001	256	[16, 32, 16, 8]	[1e-06, 1e-06, 1e-06, 1e-06]	[0, 0]	8	20	4
0.41	0.092	613.0559	0.001	256	[16, 32, 16, 8]	[1e-06, 1e-06, 1e-06, 1e-06]	[0, 0]	8	20	4

Table B.7: Top 5 grid search parameter sets ranked on validation recall@10 of BPR on Amazon 20K Users

train_time	total_val_rec	val_rec@10	nolf	n_iterations	sample_size	learning_rate	rho	sigma	reg_user	reg_item
211.0553	0.3	0.068	8	25	89 654.5	0.08	1.1	0.5	0.1	0.1
269.485	0.294	0.066	8	25	143 447.2	0.1	1.1	0.5	0.1	0.1
260.4696	0.308	0.064	8	25	143 447.2	0.12	1.1	0.5	0.1	0.1
259.8149	0.304	0.062	8	25	143 447.2	0.15	1.1	0.5	0.1	0.1
205.6254	0.294	0.062	8	25	89 654.5	0.05	1.1	0.5	0.1	0.1

Table B.8: Top 5 grid search parameter sets ranked on validation recall@10 of CFRNN on Amazon 20K Users

val_rec@10	total_val_rec	train_time	epochs	BATCH_SIZE	learning_rate	delta	max_seq_len	embedding_dim	rnn_units	pad_value
0.005	0.019	159.1642	20	32	0.1	0.2	20	100	20	90 395
0.004	0.017	160.0786	20	32	0.05	0.01	10	100	20	90 395
0.004	0.016	159.4162	20	32	0.05	0.2	20	100	20	90 395
0.004	0.019	163.2966	20	32	0.05	0.2	30	100	20	90 395
0.004	0.015	824.5462	50	64	0.2	0.2	30	100	20	90 395

Table B.9: Top 5 grid search parameter sets ranked on validation recall@10 of NeuMF on Amazon 20K Users

total_val_rec	val_rec@10	train_time	learning_rate	batch_size	layers	reg_layers	reg_mf	nolf	epochs	num_neg
0.075	0.017	293.0282	0.00005	256	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[1e-06, 1e-06]	8	20	4
0.073	0.016	155.4766	0.0001	512	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[1e-06, 1e-06]	8	20	4
0.058	0.014	306.6946	0.0001	512	[16, 32, 16, 8]	[1e-06, 1e-06, 1e-06, 1e-06]	[1e-06, 1e-06]	8	20	8
0.049	0.01	210.6117	0.00005	256	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[1e-06, 1e-06]	8	20	8
0.041	0.01	332.5113	0.00005	512	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[0, 0]	8	20	8

Table B.10: Top 5 grid search parameter sets ranked on validation recall@10 of BPR on Am-like-ML

train_time	total_val_rec	val_rec@10	nolf	n_iterations	sample_size	learning_rate	rho	sigma	reg_user	reg_item
201.5636	0.328	0.072	8	25	141 918.4	0.05	1.1	0.5	0.1	0.1
151.6832	0.278	0.07	8	25	88 699	0.05	1.1	0.5	0.1	0.1
91.85337	0.28	0.07	8	25	88 699	0.12	1.1	0.5	0.1	0.1
103.9839	0.286	0.07	8	25	53 219.4	0.1	1.1	0.5	0.1	0.1
75.9479	0.286	0.07	8	25	17 739.8	0.05	1.1	0.5	0.1	0.1

Table B.11: Top 5 grid search parameter sets ranked on validation recall@10 of CFRNN on Am-like-ML

val_rec@10	total_val_rec	train_time	epochs	BATCH_SIZE	learning_rate	delta	max_seq_len	embedding_dim	rnn_units	pad_value
0.046	0.164	130.3405	20	64	0.1	0.01	30	100	50	86 843
0.046	0.162	126.6389	20	64	0.1	0.01	10	100	20	86 843
0.044	0.176	125.9628	20	64	0.1	0.2	30	100	20	86 843
0.044	0.176	139.3441	20	32	0.05	0.01	20	100	50	86 843
0.044	0.168	414.6351	50	16	0.05	0.2	20	100	20	86 843

Table B.12: Top 5 grid search parameter sets ranked on validation recall@10 of NeuMF on Am-like-ML

total_val_rec	val_rec@10	train_time	learning_rate	batch_size	layers	reg_layers	reg_mf	nolf	epochs	num_neg
0.254	0.054	158.5695	0.00005	512	[16, 32, 16, 8]	[1e-05, 1e-05, 1e-05, 1e-05]	[0, 0]	8	20	4
0.212	0.046	151.8709	0.00005	512	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[1e-05, 1e-05]	8	20	8
0.214	0.046	168.1218	0.00005	256	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[1e-06, 1e-06]	8	20	8
0.194	0.044	540.3837	0.0001	256	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[0, 0]	8	20	8
0.198	0.044	170.349	0.00005	256	[16, 32, 16, 8]	[0.0001, 0.0001, 0.0001, 0.0001]	[1e-05, 1e-05]	8	20	8

Appendix C

Technical Environment

The testing and final runs of the algorithms were performed on a laptop provided by *YGroup* (Y) and on their Paperspace account. Paperspace is a cloud service that offers tools and computing power to developers[1]. The specifics per device used for each algorithm are shown in Table C.1.

BPR has been implemented in Python using pythons Numpy[2] package

CFRNN has been implemented in Python using TensorFlow[3] and Keras[4].

NeuMF has also been implemented in Python using TensorFlow[3] and Keras[4]. The algorithm used to create the results of X. He et al. (2017) has been used and altered. This setup can be found on *NCF Framework* (2018).

Table C.1: Specifications per device used for testing and training each algorithm

Algorithm	Device	CPU	GPU	Ram
BPR	Laptop	Intel i5-6300U (2 CPU)	-	16GB
CFRNN	Paperspace P5000	Intel Xeon (8 vCPUs)	NVIDIA Quadro P5000 (16GB)	30GB
NeuMF	Paperspace C7	Intel Xeon (12 vCPUs)	-	30GB

[1] `https://www.paperspace.com/`
[2] `https://numpy.org/`
[3] `https://tensorflow.org`
[4] `https://keras.io/`