Business Intelligence

Aarhus School of
Business & Social Sciences

Predicting E-commerce Customer Behavior using Recurrent Neural Networks & Data Augmentation Techniques

Authors:
Nick Rishoj Danmand
Mathias Smedemark Kristiansen

Supervisor:
Ana Alina Tudoran

Character count excluding spaces: 154.065
Count plus figures and tables: 175.665

June 1st 2017

Abstract

This research project improves recommendation engines of today by introducing a novel model-based approach using recurrent neural networks, a sub-class of artificial neural networks. The research was conducted using customer behavioral session data from two large e-commerce webstores located in Europe, RecSys and Audio Visionary Music. Recommender systems are essential in preserving customer interactions and attention. E-commerce enterprises gain great competition advantages when recommender systems provide customers with support in making decisions while shopping. In particular, insight into interests of new customers can be difficult to achieve due to lack of behavioral data.

With only sparse research on recommender systems built using deep learning, we provide novel artificial neural networks of high predictive power when forecasting customer behavior. This research provides recommendations of highly improved accuracy, consequently improving competitive advantages significantly by capturing the interests of new customers. Results were compared to a baseline model built using the k-nearest neighbor algorithm, a common method for generating recommendations.

Data was modeled using deep learning techniques, in particular recurrent neural networks specializing in sequential data, such as customer sessions. Recurrent neural networks currently demonstrate state-of-the-art results in natural language processing tasks, such as predicting words in sentences. We argue that sequences of words (sentences) share similar properties to sequences of customer clicks (sessions). By applying techniques known from natural language processing, this research treats customer sessions as human sentences, in order to predict the next customer move.

Two basic models were found, each with different combinations of hyperparameter values depending on the data source. Furthermore, experiments were made by augmenting data before training models. Finally, we found that recurrent neural networks outperforms a baseline model built with kNN by 41.3% (RecSys) to 161.9% (AVM), increasing accuracies from 50.65% and 20.18% to 71.55% and 52.85%, respectively. Data augmentation techniques were found to have no or only slight effect on the RecSys model, whereas data augmentation improved the prediction quality of recommendations for AVM by a few percentage points. The increase in accuracy of customer behavioral predictions will consequently improve customer loyalty and thereby revenue — as increased quality in recommendations provide customers increased quality in their basis for decision making while shopping.

Contents

List of Figures

List of Tables

List of Abbreviations and Notations

a	Scalar, i.e. an integer or a real number
$\boldsymbol{a}$	A vector of scalars
$\boldsymbol{A}$	A matrix
a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
$A_{i,j}$	Element in cell i, j of matrix $\boldsymbol{A}$
$\odot$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\approx$	Approximately / around.
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\boldsymbol{k}$**NN**	The k-nearest neighbor algorithm
NLP	Natural language processing
t-SNE	t-distributed stochastic neighbor embedding

Part I

Preface

CHAPTER 1

Introduction

> *"As of the mid-1990s the lesson has still not been learned. An 'information superhighway' is proclaimed without any concern about the traffic jams it can produce or the parking spaces it will require. Nothing in the new technology increases the number of hours in the day or the capacities of human beings to absorb information. The real design problem is not to provide more information to people but to allocate the time they have available for receiving information so that they will get only the information that is most important and relevant to the decisions they will make. The task is not to design information-distributing systems but intelligent information-filtering systems."*
>
> Simon (1996)

The rapidly growing amount of information on the internet is becoming an increasing problem (Khoshneshin and Street, 2010). Information overload can be problematic for customers, no matter the type of information sought, as ordinary people cannot consume it all. A growing number of e-commerce sites are offering over a million products and when customers seek specific genres or products, they quickly get lost in the countless options. As a consequence, customers might miss products of interest, fail to make decisions or seek products elsewhere.

Dealing with this problem and providing the right information to customers requires for businesses to take advantage of the vast amount of available data. The data generated on e-commerce sites often contain patterns that, if leveraged correctly, can be highly appreciated. However, finding and leveraging these patterns can be a complex analytical task. This report dives into the possibilities and challenges of mining customer behavioral patterns by utilizing state-of-the-art research within the area of deep learning and recommendation systems. The goal of this research is to develop a scientific machine learning model, capable of predicting future customer behavior by analyzing historical sequences of

customer clicks stored by e-commerce websites. In particular, we aim to provide personalized information about specific products that interest the individual customer. In the world of e-business, this concept is referenced as a recommendation system: recommending products to customers. Recommendation systems (or *recommender systems*) aim to understand the behavior of customers and foresee the information customers need. This, in short, may lead to higher revenues and improved customer experience in the area of e-business (Yi et al., 2014).

Recommender systems have been a highly researched field of science since the mid-1990s. To this day, this field remains equally, or even more, interesting, as the amount of stored data seems to increase exponentially (ffoulkes, 2017). The application of recommender systems is tending towards new areas, including music, movies, social media and news feeds. Naturally, this will require the ability to filter massive amounts of information, as predicted by Simon (1996).

In the area of e-business, medium- and large-scale webstores are tending to broaden their inventory categories, increasing the difficulty of showcasing products and services interesting to the individual customer. In order to increase revenue, companies show interest in promoting their products and services that may be of interest to the customer. Keeping customers in the webstore can be difficult and it is therefore important to notify the customer about potentially interesting products (Schafer et al., 2001b).

Providing customers with useful information means analyzing and understanding customers/product interactions. To date, recommender systems successfully provides interesting products to loyal and returning customers, but continue to struggle with new visitors (Hidasi et al., 2016). The most popular recommendation techniques do not work well with sparse customer data and furthermore, new customers tend to be a large proportion of the total visitors to the average webstore (Tan et al., 2016). Therefore, considering the increasing competition, it is vital to capture the attention of visitors straightaway.

While the immense amount of data can be a problem, it also, if assessed correctly, provides countless possibilities. It is plausible to define data as the concept of experience; as humans, we learn from previous experience. Often, the more we are exposed to and experience a set of challenges, the more we improve our skills in this area. This does not only apply to humans, but can be understood in machine terminology as well. Specifically, we call this *machine learning*, a subcategory of the field of *artificial intelligence*. Machine learning is a term which defines a subset of techniques that, if tuned correctly, will be able to predict some output based on input data. Recently, researchers have found that one particular set of techniques enables machines to learn complex data, such as recognizing images and human speech. They call this specific set of techniques *deep learning* techniques.

Deep learning traces back to 1943, but was not a popular topic of research until re-

cently. Until recently, the average computer did not have the power to train deep learning models, due to the amount of computations needed. Now, Google, Facebook, Tesla and other tech-leading companies uses deep learning to recognize complex patterns in data, something that was not previously achievable. For example, models aimed for image classification let Tesla cars drive automatically by recognizing the road ahead (Parloff, 2015). However, some areas of deep learning in which deep learning can provide useful insight remain less scientifically researched — such as the area of recommender systems (Goodfellow et al., 2016).

1.1 Problem Statement

In order to gain insight into customers' preferences, one may suggest surveys or other customer feedback possibilities. However, studies show that direct customer feedback tends to be sparse. Instead, patterns of customer visits hold data that can be interpreted as customers' subjective ratings (Yi et al., 2014).

Not every customer has a historical relationship with webstores, such as new customers. A large issue regarding new customers lies in the fact that little or no previous information is known. Data gathering happens as customers begin their journey on a webstore. However, the amount of data needed for recommendation systems tends to stretch over multiple products, requiring loyal customers. This problem, known as the *cold-start* problem[1], decreases the chance of recommending any valuable information in time, before the new customers leave the site (Schafer et al., 2001b).

1.1.1 Research Questions

i) How can artificial neural networks infer consumer preferences and subsequently attract and retain consumers?

ii) How can artificial neural networks be trained upon session-based data structures?

iii) To what extend can artificial neural networks improve performance of today's recommender systems?

The final objective of the thesis is to increase sales of e-commerce enterprises by improving the consumer's shopping experience with relevant recommendations based on historical consumer behavior. Aiming at medium- and large scale e-commerce businesses, we seek to mine frequent patterns in historical session data and, overcome problems of data sparsity and cold-starts.

[1] In depth explanation is provided in section 3.3.

1.2 Delimitation

The following points discuss the delimitations of this research.

- The subject of deep learning is extensive and in many ways complex. While we assume the reader to have some basic general understanding of predictive methods, we aim to cover the basics of artificial neural networks. These networks will be discussed to a degree that allows the reader a deeper understanding of how this research utilizes artificial neural networks. We will be covering the core technical aspects of artificial neural networks. The mathematics behind the inner working of these networks are discussed to a degree, that should provide the reader with an insight into the advantages and disadvantages of the use of deep learning techniques. While we describe technical details of artificial neural networks, we emphasize that the theoretical framework of this thesis remains brief, considering the complex topic in itself.

- Artificial neural networks come in a wide range of types and architectures, each with its strengths and weaknesses. This thesis utilizes recurrent neural networks. Other types of artificial neural networks will not be covered, since their strength lies in other data structures.

- Collaborative filtering techniques will be described in depth, while content-based filtering will not. Content-based filtering uses another type of data structure to generate recommendations that is not relevant for this research.

- The models behind the research project were built in plain python using Google Tensorflow — all included in the appendices. However, the thesis itself omits any direct discussion of source code, as we chose to focus on the architecture of models, tuning models, model performances and outcomes.

1.3 Contributions

We aim to improve the customer decision basis for e-commerce webstores by developing new, state-of-the-art recommendation systems based on recent research in deep learning techniques. As a large number of recommender systems today struggle to hold the attention of new customers, we contribute with models that overcome this problem. In this section, we briefly present the main contributions of this thesis.

Source of Data We propose to analyze and use click-stream data for recommendations. As many webstore users are new users, we only work with data provided by new

users, e.g. items they are looking at (Tan et al., 2016). By only focusing on sessions, we capture the new users' activity. By analyzing and modeling only this data, we can improve the quality of recommendations to new users.

Improvements in Business We contribute with models that improve the quality of recommendations. Based on the goals of a recommender systems, we propose models that make quick, relevant, diverse recommendations based on simple session data that is available for most (in theory, all) webstores. We argue that these improved recommendations increase the users' visit times and conversion rates. Hence, webstores could gain higher revenues and thus higher profits by promptly getting the attention of new, interested users.

Data Augmentation Besides modeling session data, we contribute with multiple data augmentation techniques to improve the accuracy of the models in question. Data augmentation techniques can ultimately help mine the frequent patterns in the session data, consequently improving the performance of models. For example, reversing session data can increase the quality of recommendations for e-commerce webstores with sparse session data.

Deep Learning We introduce a novel field of research. The application of recurrent neural networks on session data was first introduced by Hidasi et al. (2016) and later improved by Tan et al. (2016). We aim to improve these models by restructuring the deep learning architectures and applying data augmentation techniques. The models were built using Google Tensorflow and, to the best of our knowledge, this is the first recurrent neural network recommender system developed with Tensorflow. To evaluate the performance, we apply the models on two datasets, one of which was used by Hidasi et al. (2016) and Tan et al. (2016). Recurrent neural networks are widely applied in natural language processing (NLP)[2]. Since human sentences have similarities with online human behavior (sessions), most of the techniques applied in this research came from the area of NLP. Chapter 4 discusses the similarity between sentences and sessions in depth. This research contributes by utilizing these NLP techniques on sequential session data.

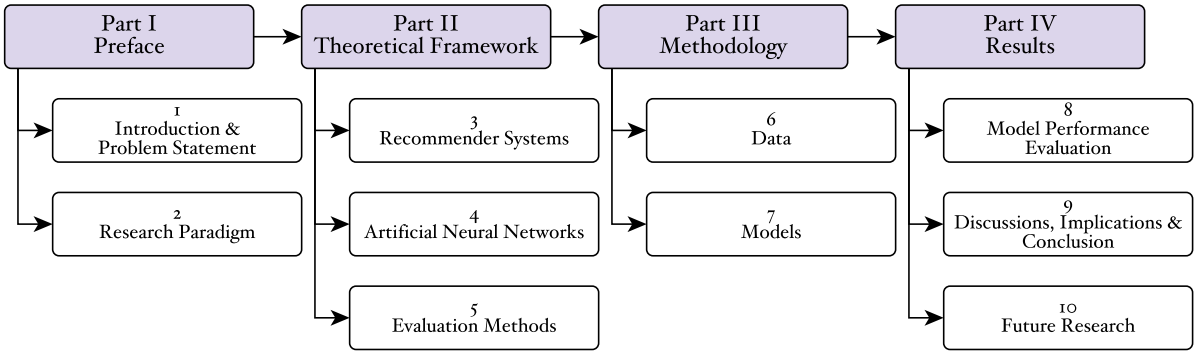
Big Data Working with millions of rows of data is another motivation for utilizing NLP techniques to tackle large catalogs of products that recommender systems often struggle with[3]. By introducing word embedding, a technique known from NLP, we contribute by improving the computation time and transforming items into computable vectors, making it possible to work with thousands or even millions of items.

[2]NLP is a set of techniques capable of capturing sentiment and classes of texts.

[3]In NLP, for example, words of a book can be considered the product catalog, consequently making the catalog of unique words large.

1.4 Structure of Thesis

This research project is written in four parts. Within each part, chapters and sections exist. The end of each chapter includes a short summary with important takeaways. Figure 1.1 illustrates the chapters of the four parts.



Source: Own elaboration.
Figure 1.1: Structure of the Thesis.

Part I: Preface Chapters 1 and 2 cover the background of the thesis by introducing the problem, contributions, delimitations and research paradigm.

Part II: Theoretical Framework Chapters 3, 4 and 5 discusses the theoretical framework used to support the work of the thesis. Chapter 3 gives the reader an understanding of recommender systems and the implications of developing good recommendations. Chapter 4 presents the theoretical groundwork of artificial neural networks and recurrent neural networks. Chapter 5 describes how we evaluated and validated the performance of the models proposed in Part III.

Part III: Methodology Chapter 6 digs into data preprocessing of the datasets as well as data augmentation that could improve data quality. Chapter 7 provides justification for the architectures and tuning of the models.

Part IV: Findings Chapters 8, 9 and 10 show how the models perform and conclude on findings. Chapter 8 compares the performance of the models with different data augmentation techniques. These findings are discussed and concluded on in chapter 9 and chapter 10 reflects on future research.

CHAPTER 2

Research Paradigm

A paradigm is the belief system that consists of a specific set of concepts, theory, postulates and research methods that determine how researchers do and perceive concepts. Research paradigms can be characterized by distinctions: Ontology, Epistemology, and Methodology. These characteristics determine how researchers view knowledge and reality (Guba, 1990).

The thesis is based on the research paradigm named post-positivism, where deductive logic is applied to support the creation of knowledge. Post-positivism is a modified version of positivism, acknowledging that observations are prone to errors. The goal of positivism is to uncover the truth, whereas post-positivism acknowledges that truth is impossible to accurately perceive; it can only be approximated (Creswell, 2013).

Post-positivism also has a wider criteria for data acceptance, making this paradigm a favorable fit for this research. This research seeks to approximate the truth by assessing rich data sets (Creswell, 2013).

Ontology looks at what reality is, and the ontological aspect of post-positivism has a focus on critical realism. Post-positivism acknowledges that a reality exists by natural causes, but this remains impossible to accurately perceive because of fallible data. Therefore, the ultimate truth remains unknown and observations remain fallible and error-prone (Guba, 1990). When assessing the data supporting this thesis, a critical view on the data and its imperfections is necessary. Another important aspect within post-positivism describes critical tradition: whether the findings of this thesis fit existing knowledge. Previous findings supporting this thesis will be references.

Epistemology concerns the understanding of knowledge creation, knowledge justification and the rationality of belief. Post-positivism advocates modified objectivity, meaning that even though objectivity remains as the ideal, it can only be approximated. A close to ideal objectivity can be archived by remaining as neutral as possible. Post-positivism also emphasizes the importance of external verification and multiple measures and observations (Guba, 1990).

The methodology within post-positivism focuses on modified experiments and advo-

cates the use of multiple methods to support the research (Creswell, 2013). Hence, multiple models are tested to support the performance. The observation in post-positivism cannot be fully trusted, due to errors. These errors together with the emphasis on multiple external verifications is the reason for multiple sources. Using multiple sources of data can decrease these errors and give a more probabilistic view of reality. Furthermore, post-positivism allows many imbalances to gaining research that are both realistic and objective, which will result in a tradeoff between internal and external validity (Guba, 1990).

Part II

Theoretical Framework

CHAPTER 3

Recommender Systems

"We have 6.2 million customers; we should have 6.2 million stores. There should be the optimum store for each and every customer."

Jeff Bezos, CEO of Amazon.com (BusinessWeek, 2003)

Medium and large-scale e-commerce webstores may have product repertoires so large, that it seems overwhelming and unmanageable for any customer. The item selection process for a new visitor can become complicated and frustrating (Kunaver and Pozrl, 2017).

Recommender systems play an essential role in the e-commerce market competition. Historically, we, as consumers, would receive recommendations from friends, experts, magazines or newspapers. These sources all suffer from certain limitations in discovering new, relevant offerings. Therefore, recommender systems now serve as a technological proxy for the historical social recommendations, by predicting and recommending products any customer may find interesting based on historical customer data. These personalized recommendations therefore improve the shopping experience while providing useful products to the customers (Ekstrand et al., 2010; Hidasi et al., 2016).

According to Ricci et al. (2011), improving user experience can affect e-commerce sales in three different ways:

1. Converting the browsing customer into a buying customer by quickly providing useful information, such as the product they were — or did not know they were — looking for.

2. Increasing cross-selling by suggesting additional and relevant products pre, during or post transaction.

3. Building customer loyalty by providing personalized experiences.

Ultimately, recommendation engines utilize data in order to serve products of relevance to the customer. In the terminology of recommendation systems, we call the customer entity an *user* and product for an *item* (Aggarwal, 2016). Recommender systems work by analyzing the historical interaction that happens between users and items. Examples of interactions would be any user visiting, purchasing or rating an item. By finding correlations between users and items, recommender systems predict the probability of a user being interested in an item (Kunaver and Pozrl, 2017).

In almost three decades, recommender systems have been interesting to researchers and still remain an expanding field of research. Today, recommender systems are found in almost any type of application. However, not all recommender systems are alike. In fact, they tend to differ, as multiple types of recommendation method have been developed since the birth of recommender systems. Two widely-applied types of recommender systems within e-commerce are content-based filtering and collaborative filtering (Ekstrand et al., 2010).

Content-based Filtering The content-based approach works by analyzing the item-to-item relationship; it recommends items that have similar item attributes and thereby avoids cold-start[1] problems. This, however, tends to become problematic with large item sets and unstructured item attributes (Ekstrand et al., 2010).

Collaborative Filtering This approach recommends items based on similarities between the users' behaviors. It relies heavily on the fact that users' past preferences tend to also be future preferences. A great advantage of collaborative filtering is that it does not rely on item content or attributes (Aggarwal, 2016).

In this thesis, we apply deep learning techniques to session data to predict the behavior of users, hence our focus remains on collaborative filtering to analyze and predict user behavior.

User behavior tends to be complicated and difficult to predict with sparse data on each user. Traditionally, recommender systems rely on the data of user profiles. This requires data about previous sessions of logged-in users. In a perfect world, everyone would log in everywhere. Sadly, this is rarely the case for users of e-commerce sites. This means that recognizing users become more difficult, resulting in cold-start problems for the recommender systems (Hidasi et al., 2016; Ricci et al., 2011).

To provide recommendations, data about user preferences is essential, as recommender systems analyze and try to understand the interactions between users and items. According to Aggarwal (2016), user feedback collection happens in two ways:

Explicit Feedback By explicitly asking the user to give feedback about an item. This

[1]In depth explanation is provided in section 3.3.

is usually achieved with rating systems or surveys where the user expresses opinions about items.

Implicit Feedback The second way is implicitly tracking a user's activity on a website. Tracking may include items visited, items added to cart, items purchased, time spent on site and more. This method tends to be more user-friendly than that of the explicit method, hence nothing is actively required from visiting users. However, this may lead to less accurate data on the user's actual preferences.

3.1 Collaborative Filtering

Collaborative filtering differs from other recommendation methods in the sense that no item information is used when searching for patterns. Collaborative filtering rather focuses on users' preferences towards items (Aggarwal, 2016).

Rich session data allows for utilizing techniques that can search for patterns in the session data; patterns that, if extracted correctly, may lead to recommendations. Users with similar session paths tend to have similar preferences as well. It is these similarity patterns a recommendation engine uses when producing recommendations. An assumption says that if any one of the users preferred an item, it is likely that a user with a similar path prefers this item as well (Su and Khoshgoftaar, 2009).

3.1.1 Workflow

The most typical workflow of a collaborative filtering recommendation system consists of three parts. First, a user expresses preferences towards items by explicitly or implicitly giving feedback. Second, the recommendation system searches for other items that similar users prefer. Last, the system recommends items to the user that similar users tend to prefer (Su and Khoshgoftaar, 2009).

The properties of collaborative filtering make it possible to create superior systems compared to systems built with content-based filtering. In fact, it is possible to recommend items based on complex relationships, such as taste or quality (Su and Khoshgoftaar, 2009). However, collaborative filtering is not without its drawbacks. The method also has its disadvantages as presented in section 3.3.

3.1.2 Applied Approaches

Collaborative filtering comes in two types: the memory-based method and the model-based method. The memory-based approach bases recommendations on the items rated by users. Ratings determine similarities and compute imputations for unrated items.

However, this approach has limitations and researchers eventually came up with the model-based approach. The model-based approach uses the item-user interactions to train models capable of predicting recommendations (Su and Khoshgoftaar, 2009). The following sections briefly describe the types of memory and model-based approaches. Sections 5.3 and 7 go into detail about utilizing some of the methods illustrated below.

3.1.2.1 Memory-based Collaborative Filtering

Memory-based collaborative filtering uses the entire user-item matrix database[2] to predict a user's interests in an item based on the user's previous ratings of items. In memory-based approaches, we distinguish between item-based or user-based approaches to find similarity patterns. Similarities can be calculated between users or between items based on paths of users. This thesis applies the item-based approach by grouping items based on similar ratings given by users (Su and Khoshgoftaar, 2009).

Memory-based recommender systems have their advantages in simple matrix computations and intuitive implementations, as well as in intelligible recommendations. Difficulties for these systems typically lie in sparse data issues and cold-start problems. However, the computations needed to group items or users increases exponentially as the number of items or users increases — as the user-item matrix grows. Therefore, the computation time needed to compute the user-item matrix on large data can be problematic (Aggarwal, 2016).

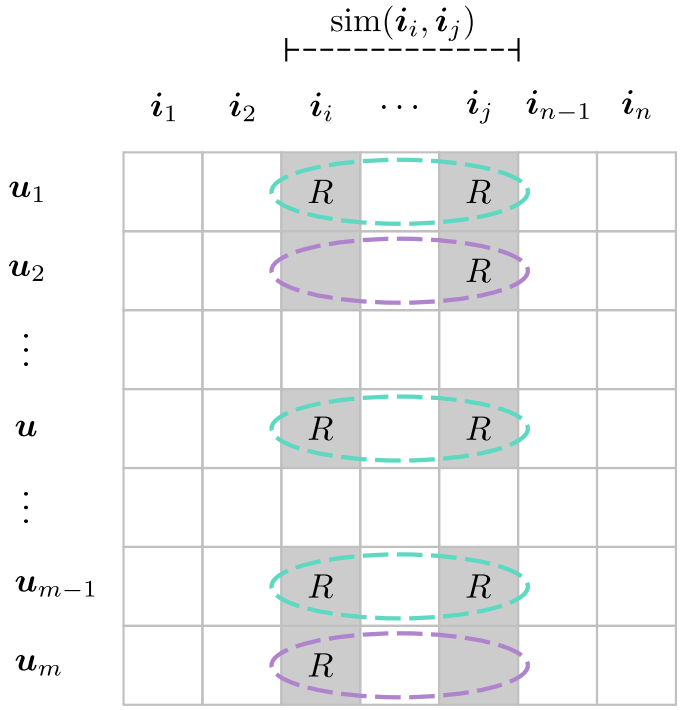
A popular method of memory-based recommendations is the k-nearest neighbor algorithm (kNN). kNN computes the distance between items based on any user's preferences. By analyzing user ratings, kNN assigns items to a class membership of similar items. When computing the predictions, only user-rated items are considered. Items that frequently show the same ratings are grouped as neighbors; if a user rates an item from one neighborhood, the recommendations for this user will presumably be from the same neighborhood (Sarwar et al., 2001).

As mentioned earlier, the kNN approach can struggle with large data due to an accordingly large user-item matrix. Furthermore, large data tends to be sparse, making it difficult for the algorithm to compute predictions (Schafer et al., 2001a). kNN will serve as a baseline model[3] during this project.

To compute item similarities, kNN compares the behavior or ratings of users. Figure 3.1 illustrates an example of a user-item matrix with item ratings vectors $\boldsymbol{i}_i$ and $\boldsymbol{i}_j$ isolated. User ratings are marked gray. User u_1 has rated the items i_i and i_j, which may indicate that i_i and i_j share similarities in terms of ratings R. However, user u_2 has only

[2]Basically a matrix holding items on one axis and users on the other, usually stored in databases.

[3]We define a basic baseline model based on a popular algorithm for recommendations. Models developed throughout this research will compare performances with the baseline model.



Source: Own elaboration.

Figure 3.1: How k-Nearest Neighbor Computes the Cosine Distance.

rated item i_j which indicates the opposite, thus i_j seems distant from i_i. Having numerous users rating both items makes rating vectors $\boldsymbol{i}_i$ and $\boldsymbol{i}_j$ more alike, increasing the chance of items i_i and i_j being neighbors and vice versa.

kNN works with a similarity (or distance) algorithm, such as the cosine distance. The algorithm calculates similarity between item vectors holding all user ratings, or missing values if no rating has been given (Sarwar et al., 2001). Specifically, the similarity between $\boldsymbol{i}_i$ and $\boldsymbol{i}_j$ is given by:

$$\text{sim}(\boldsymbol{i}_i, \boldsymbol{i}_j) = \cos(\boldsymbol{i}_i, \boldsymbol{i}_j) = \frac{\boldsymbol{i}_i \boldsymbol{i}_j}{||\boldsymbol{i}_i||\,||\boldsymbol{i}_j||}$$

Cosine similarity takes the number of co-occurrences of two items in sessions and divides this by the square root of the product of the session in which the individual items occur. The equation produces a number between zero and one, representing the similarity between items. A value of one means they are similar while zero shows that items are distant (Tan et al., 2005).

Cosine similarity will be the algorithm applied in the baseline model.

3.1.2.2 Model-based Collaborative Filtering

The model-based approaches are based on predictive machine learning models. Building and training some of these models can take hours or even days to complete. In the training process, machine learning algorithms learn to recognize patterns used for making

recommendations (Su and Khoshgoftaar, 2009).

This approach seeks to uncover latent factors that can explain the observed preferences of users. The model-based approaches overcome some of the problems with memory-based methods like sparsity and large datasets. However, models can be expensive to build and often tend to be black boxes. A black box is a system whose inner workings can be difficult to fully understand. Black box systems often take some input and deliver some output — while observers have no or little knowledge of the inner workings of the box.

Chapter 7 elaborates on the model-based algorithms in this research project; in particular, we utilize artificial neural networks.

3.2 Goals of Recommender Systems

As previously discussed, Ricci et al. (2011) present the overall goals of using recommender systems: to increase revenue by supporting customers and to personalize customer experiences. By recommending interesting and relevant items to the customers, fewer customers get lost in information. To achieve higher revenue, Ricci et al. (2011) present the technical and operational goals:

Relevance A recommender system should present the user with relevant user-specific items, increasing the chance of conversions[4]. However, there are a lot of challenges when building recommender systems. Relevance tends to be important, but more factors play a major role regarding the quality of the systems.

Novelty The recommended items should be something that the user has not seen in the past, as repeated recommendations may lead to reductions in sale diversity.

Serendipity The users should be presented with unexpected but relevant items. This may lead to new interests for users and thereby help increase sale diversity. By incorporating the possibility of items of serendipity, new interest can be created for the user and this can increase sale diversity. However, it can lead to irrelevant results as well. Serendipity should therefore be introduced with caution and be accurate in its recommendations before the advantages outweigh the disadvantages.

Diversity When recommending items, items should be diverse, giving a user multiple choices. If the systems show four similar items, the risk of users disliking all items increases. Showing diverse collections of items has been proven to increase the possibility of interested users.

[4]In e-business, a conversion is a transaction between customer and store.

3.3 Challenges with Collaborative Filtering

All recommender systems face challenges. Recommendations tend to be imperfect, as data based on human behavior can be sparse or incomplete (Meyer, 2006). Developing and designing good and robust systems requires the addressing of such various challenges.

Cold-start Recommender systems easily suffer from sparse data; data with only few ratings. Because users tend to visit or rank only a small fraction of the items, it becomes difficult for the recommender systems to recommend interesting items. As a result, new recommendations for new users suffer since no data on new users' preferences is available. Most recommender systems need a portion of data in order to serve useful and precise recommendations. Similar problems arise when a new item is introduced since new items hold no history of user preferences (Meyer, 2006; Ricci et al., 2011).

This is why content-based recommender systems are sometimes preferred, as they are robust to the cold-start problem and therefore perform better when knowledge about users is sparse. However, content-based recommendations do not account for user behavior and often recommend irrelevant items (Khusro et al., 2016).

Scalability The scalability of the recommender system is a key feature for real-world datasets. A recommender system may perform well on small datasets, but tends to become inefficient and make poor recommendations when introduced to a large dataset (Meyer, 2006; Ricci et al., 2011).

Furthermore, ensuring a real-world suitable recommender system means accounting for computation time and the time it takes to make live recommendations (Meyer, 2006).

Reactivity Tracking changes to user profiles and preferences in real time is important. If a recommender system does not adjust to preferences in a real-time session, the recommendation can get stuck showing outdated items. This can be an irritation factor for users leading to trust issues towards the site (Sarwar et al., 2001).

Temporal Changes Time in general can play a big role in recommendation engines. Item preferences may evolve over time and the popularity of items will vary. The importance of temporal changes can vary depending on the website. Items such as movies, fashion and interior may have high temporal changes for user preferences (Ricci et al., 2011).

Popularity Bias & Diversity When a recommender system makes recommendations, problems arise as systems tend to recommend items of high popularity — which

may be a narrow range of items. This can eventually lead to recommender systems becoming worse over time, as recommending popular items makes popular items seem even more popular. If this happens, the range of recommendation probabilities often shrink. Recommendation engines therefore need to adjust for overfitting or popularity biases (Meyer, 2006).

Collaborative filtering methods have a challenge in the fact that only items that have received feedback will be considered when computing recommendations. When the number of items is large, a long tail[5] of items may never be visited. Consequently, these items will not be considered when computing recommendations. To overcome this challenge of sparse data, recommender systems need to account and incorporate all products (Meyer, 2006).

Shilling Attacks Malicious users or competitors are known to affect recommendation systems by increasing or decreasing items' popularity. Shilling attacks can decrease the performance and quality of a recommender system, which will break the trust of users. Collaborative filtering methods are affected much more by these attacks than content-based. The consequences of such attacks should be discussed and countermeasures should be planned (Khusro et al., 2016; Ricci et al., 2011).

3.4 Summary

In the world of e-commerce, recommender systems are used to make recommendations for customers visiting webstores. Ultimately, this will help customers in decision making and in search of products. This chapter begins by presenting two main types of recommender systems: the content-based filtering method and the collaborative filtering method. Content-based filtering includes recommending products based on information about products such as color and size. Collaborative filtering focuses on similarities in users' behavior. Collaborative filtering suffers from various challenges. The motivations behind the collaborative method lie in the fact that rich recommendations can be made based on customers' behavior.

This chapter also present the baseline model of this research, a k-nearest neighbor model. kNN can be utilized as a memory-based collaborative algorithm. Another collaborative filtering approach presented is the model-based approach. This approach includes training machine learning models to recognize patterns in data which can help in predicting recommendations. While models can serve good recommendations if trained correctly, models can be less intuitive and sometimes tend to be a black box.

[5]The long tail is the part of items that are rarely seen.

To achieve a better recommender system, it should account for the four goals; relevance, novelty, diversity, and serendipity. We present most challenges with collaborative filtering methods, such as temporal changes, diversity and the cold-start problem. These challenges should be assessed when considering developing a collaborative filtering recommendation system.

CHAPTER 4

Artificial Neural Networks

> *"The field of artificial intelligence, or AI, attempts to understand intelligent entities. Thus, one reason to study it is to learn more about ourselves."*
>
> Russell and Norvig (2003)

Simple machine learning techniques, such as decision trees, support vector machines and the unsupervised k-means clustering, have yet to succeed in solving AI problems, such as recognizing speech or recognizing objects. Recently, deep learning has become successful in modeling such AI tasks. This chapter presents deep learning techniques that may assist in modeling online human behavior, hence, as expressed by Russell and Norvig (2003), to "learn more about ourselves".

4.1 Background

'Deep learning' is a phrase used to describe artificial neural networks that consist of more than two layers. Starting from the beginning, Rosenblatt (1958) presented the first article on training a single (artificial) neuron back in 1958. In the mid-1980s, back-propagation was introduced, an essential ingredient of current deep learning theory (Bishop, 1995; Rumelhart et al., 1986). Around the year 2000, when computers became powerful enough to train artificial neural networks, the topic of deep learning became much more popular (Marr, 2016).

Artificial neural networks consist of multiple layers and each layer consists of multiple *neurons* (we call them *nodes*). In neuroscience, neurons take inputs from other neurons and "compute" (in fact, they *activate*) an output based on these inputs. In artificial neural networks, the concept is the same. Between each layer of an artificial neural network, nodes receive some input from one or more nodes in previous layers and finally, compute some output based on the inputs.

A feedforward neural network is the basic form of an artificial neural network. This form of network estimates some function f, by learning the values of its parameters. Layers of a network can be considered functions that map an input to an output through the nodes of the layers. The reason feedforward neural networks are named 'networks', is due to the combination of functions they consist of. As an example, a neural network could be written as $f(x) = f_3(f_2(f_1(x)))$, where f_1 is the first (input) layer, f_2 is the second layer and f_3 is the third layer. In this case, f_3 is called the output layer. The number of layers between first and last layer of a network determines the *depth* of the network. Layers in between the input layer and the output layer are called hidden layers; outputs of hidden layers are never shown. Any artificial neural network with one or more hidden layers is considered as a deep neural network, hence the term *deep learning* (Goodfellow et al., 2016).

4.2 Training Techniques of Artificial Neural Networks

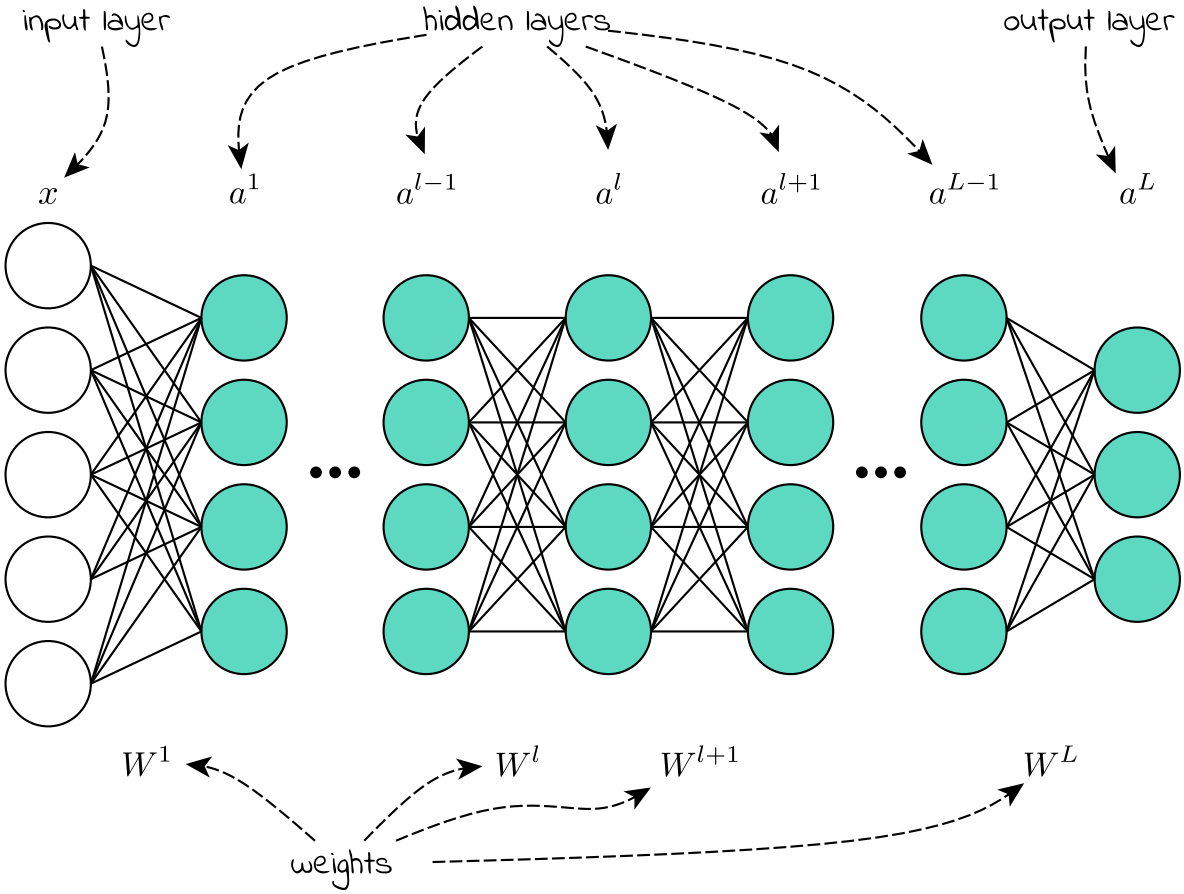
An artificial neural network consists of at least two layers; the input and output layers. Between the input and output layers, hidden layers can exist. Figure 4.1 shows a simple structure of an artificial neural network with multiple hidden layers. The input layer consists of five inputs (input nodes) and the output layer consists of three output nodes. In theory, layers of a network can hold any number of nodes (Nielsen, 2015).

Inputs and nodes are illustrated with a circle and edges[1] showing the connections between layers. The training process begins with the network receiving inputs, then information propagates forward though the network to produce some output based on the input. In figure 4.1, each node produces an output referred to as an *activation*. Activations follow the edges (also known as weights) to the next nodes. The final output of the model is often referred to as the predictions of the network. Once the final output has been produced, a loss of the network can be calculated based on a loss function that measures the distance between the predicted output and the actual targets. In order to train an artificial neural network, we want to minimize this loss function to get the minimal distance between outputs and targets. By using gradient descent, it is possible to adjust the weights (illustrated with edges) of the network by propagating backwards through the network and thereby minimize the loss (Goodfellow et al., 2016; Nielsen, 2015).

Generally, three steps are taken at each training iteration of a neural network.

1. Training data is fed into the neural network. The data propagates forwards, layer by layer, until the production of some output at the final output layer.

[1]The lines between nodes shown in figure 4.1.



Source: Own elaboration.

Figure 4.1: Basic structure of an artificial neural network.

2. The distance between the output predicted by the network and the actual output targets is calculated and called the *error* of the network.

3. Propagating backwards through the network helps determine how much each weight contributes to the total error. Weights are updated accordingly to their error contributions. In theory, the next forward propagation would therefore compute a smaller error.

Once the cycle is over, one training iteration has been completed and the iteration starts over by feeding the next batch of training data into the network. This way, weights are adjusted and the error is reduced at each training iteration.

We have now — on a very basic level — covered the most elementary steps of training a neural network. The following sections walk through the mathematical properties of training the basic neural network. This will introduce a general understanding of the mechanisms that can be applied and adjusted to train neural networks in useful ways, such as analyzing patterns in sequences of customer behavior.

4.2.1 Optimization Techniques

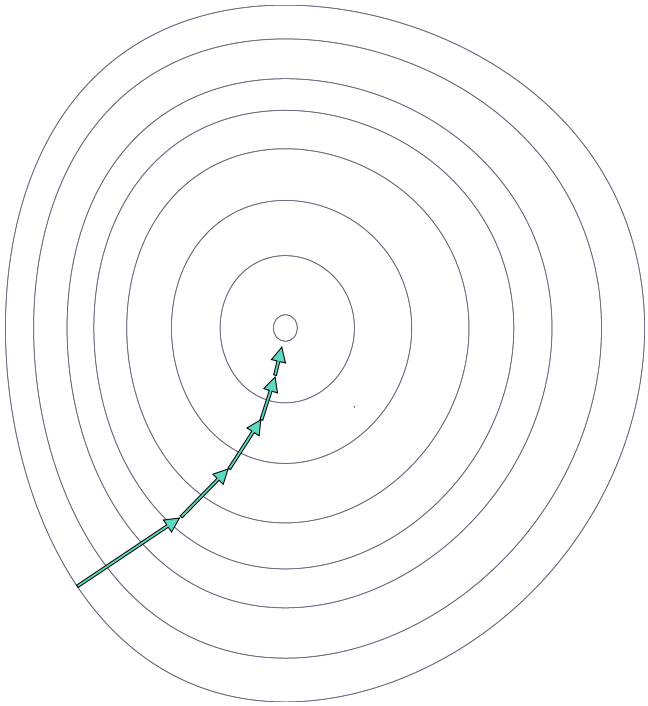
Machine learning typically applies some optimization algorithm that enables the desired model to optimize its estimates during training. To measure the goodness of the model, we use something we call a cost function[2], C. The cost function C can vary between network architectures, depending on the desired model output. The goal of the network will be to minimize C as a function of weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ (Goodfellow et al., 2016).

4.2.1.1 Gradient-Based Optimization

Machine learning generally includes minimizing some objective function. When minimizing this function, we call the function the cost or loss function as just presented. Gradient descent is a powerful tool when it comes to minimizing a cost function. Using the same example as in section 4.1, we have a function $y = f(x)$, that we wish to minimize. We know from basic calculus, that the derivative of $f'(x)$ gives the slope of $f(x)$ at the point x. The derivative therefore suggests in which direction to change x in order to make a tiny improvement in y. Using this knowledge, we can repeatedly take small steps in the suggested direction[3], approaching a lower point and thereby improving y as much as possible. Figure 4.2 illustrate how gradient descent approaches the minimum as x changes. The arrows illustrates steps towards the lower point (Goodfellow et al., 2016).

Gradient descent can be understood much like moving downhill in a hilly landscape. The goal is to reach the bottom by taking small steps and evaluating whether the step

[2]Cost functions are also referred to as loss or objective functions.
[3]As suggested by the slope.



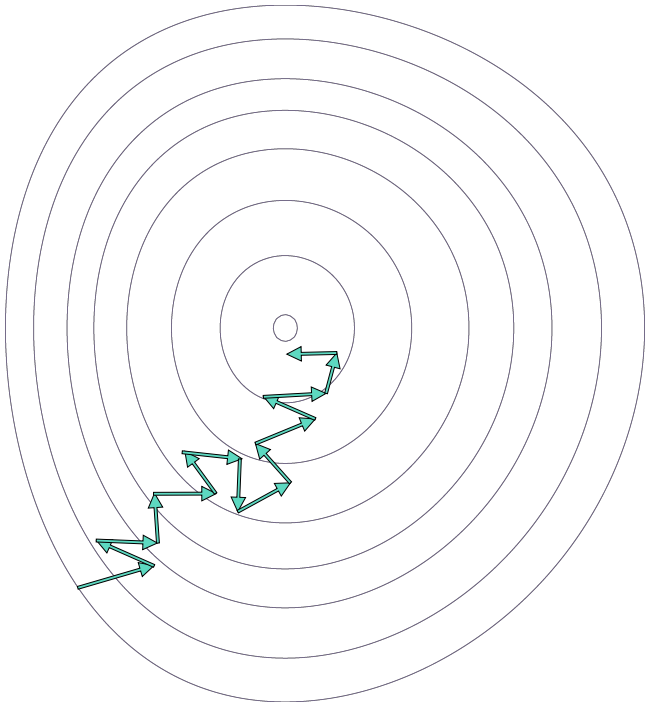
Source: Partly own elaboration - dimensional background borrowed from Wikipedia.
Figure 4.2: Gradient descent illustrated in two dimensional-space.

has increased or decreased the magnitude. By repeating these steps, we may reach the bottom of the landscape, called the global minimum. A global minimum is what we would ideally like to reach, since $f(x)$ has the lowest possible value at this point. However, while trying to find the global minimum, one may encounter what is called a local minimum. Local minimums are points where $f(x)$ is lower than the neighboring points, making it impossible to decrease $f(x)$ by taking infinitesimal steps.

This method of decreasing a cost is called gradient descent and is an elementary part of most deep neural networks. Using this approach, we can minimize the cost function by optimizing (updating) x and thereby train the model; the model is learning, hence the phrases machine learning and deep learning. In order to create a useful model, we need to apply an extension of gradient optimization.

4.2.1.2 Stochastic Gradient Descent

Stochastic gradient descent is an extension of gradient descent. When applying basic gradient descent to large datasets, the computation consumption required expands quickly, as all samples are computed upon at once. Instead, stochastic gradient descent picks a smaller number of samples in a batch, often referred to as a mini-batch. The expected gradient can be computed based on these smaller batches of samples. Samples are picked randomly (*stochastically*) from the training set. Since the batch size is smaller than the total amount of samples, we may not compute the overall correct gradient, but in



Source: Partly own elaboration - dimensional background borrowed from Wikipedia.
Figure 4.3: Stochastic gradient descent illustrated in two-dimensional space.

the long term, stochastic gradient descent takes us in the right direction using much less computation power. As illustrated in figure 4.3, it shows that each step does not approach the minimum directly, but using enough small steps takes us just as far as basic gradient descent with less energy. This way, we are able to fit training sets containing several million samples using mini-batches of samples with much less computation power compared to that of basic gradient descent. Notice that in figure 4.3, each step is not pointing directly at the minimum but, in the long run, we approach the minimum.

This research will not be applying the basic version of stochastic gradient descent, but rather a recently developed optimization method that uses stochastic gradient descent as part of its algorithm. This optimization algorithm from 2014 was developed by Kingma and Ba (2014) and is called ADAM: A Method for Stochastic Optimization. We will not cover ADAM in depth, as the technical complexity is beyond the scope of this research. However, we briefly discuss the advantages of the algorithm when applying it in section 7.2.1.2.

4.2.2 Forward Propagation

When an input, often as a vector of scalars, is introduced to a neural network, it is propagated forward in the network until the final layer is reached. In between each layer, the network applies weights to the input of nodes. In figure 4.1, edges illustrate weights between the nodes, whereas, for simplicity, biases are omitted from the figure. However,

each node is assigned a bias term[4], b. A node's output is called the *activation of a node* and refers to the inputs of nodes of the next layer. The weighted input of a single node, j, in layer l, receiving input from multiple nodes can be calculated by

$$z_j^l = \sum_k^1 (w_{j,k}^l a_k^{l-1}) + b_j^l, \tag{4.1}$$

where z_j^l is the weighted net input of node j at layer l, with the associated weights, $w_{j,k}$, connecting layer $l-1$ with layer l. a_k^{l-1} represents activations of the previous nodes at layer $l-1$. Finally, a weight class called the bias, b_j^l, is added (Nielsen, 2015).

In short, for a single node's weighted input, the products of weights and activations are computed and a bias is added. In order to generate the activation value, an activation function is applied to z_j^l. Activation functions take inputs and squashes them into activations. The sigmoid function is widely applied in neural networks; see section 4.2.4 for a detailed description. For now, we illustrate this step by applying the activation function to the weighted input:

$$a_j^l = \sigma\left(\sum_k^1 w_{j,k}^l a_k^{l-1} + b_j^l\right), \tag{4.2}$$

where a_j^l is the activation of the node j at layer l and σ is the chosen activation function. Once all steps have been taken, the forward propagation is done and the network produces an output $\hat{y}$. When training the network, $\hat{y}$ is compared to the actual y. The difference between $\hat{y}$ and y is used to calculate the loss of the network. Loss can be calculated in several ways depending on the loss function at hand.

4.2.3 Backward Propagation

Backward (or, 'back') propagation, as first presented by Rumelhart et al. (1986), uses the cost computed by forward propagation and the optimizer to adjust weights and biases at each training interval in order to minimize some cost, C. This is where the learning actually takes place in a neural network. Back propagation provides the mathematical tools to update weights and biases. This section explores the four fundamental equations behind back propagation that can be applied to any artificial neural network.

The goal is to compute the partial derivatives (referred to as gradients) $\partial C/\partial w_{j,k}^l$ and $\partial C/\partial b_j^l$ of C with respect to the weights w and the biases b. The reason partial derivatives of the weights and biases are interesting lies in their effect on the weighted inputs and the activations. We know from before that the goal is to adjust these weights and biases in such a way that we decrease the final error of the cost function. The gradients $\partial C/\partial w_{j,k}^l$ and $\partial C/\partial b_j^l$ provide an indication of how much each weight, $w_{j,k}^l$ and b_j^l, affects the final

[4]Biases are usually initiated at random and adjusted as training occurs.

cost, C. This is key knowledge when weights are updated (Bishop, 1995; Nielsen, 2015).

First, we first introduce a definition of the error of any node j in any layer l:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}. \tag{4.3}$$

The vector version of the errors at layer l is introduced as $\boldsymbol{\delta}^l$. We can now point to any error of the full network at node or layer level. This leads us to the first equation of the back propagation algorithm. The following equation (4.4) gives the error in the output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L), \tag{4.4}$$

where $\partial C/\partial a_j^L$ is the partial derivate of C with respect to the activation of node k at the final layer L. This is an error signal of how much the final cost is changing as a function of the j^{th} output activation. This also tells us that, if there is a great independence between the cost and the output of node j, then the error δ_j^L will be small. Finally, $\sigma'(z_j^L)$ is a signal of how much the applied activation function σ changes at z_j^L. Note that the form of $\partial C/\partial a_j^L$ depends on the chosen cost function. Equation 4.4 is based on a single node. In practice, we are working with vectors and matrix multiplications. Equation 4.4 can be rewritten as a matrix-based equation:

$$\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}} C \odot \sigma'(\boldsymbol{z}^L), \tag{4.5}$$

where $\nabla_{\boldsymbol{a}} C$ is a vector consisting of gradients of C with respect to activations a_j^L. Note that $\odot$ is the Hadamard[5] product of the two matrices (element-wise multiplication) (Nielsen, 2015).

Now, we are able to compute the errors of the final layer L. Assume we propagate back one layer, $L-1$, and reach a layer l. The following second back propagation equation (4.6) computes the error $\boldsymbol{\delta}^l$ of this layer:

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l), \tag{4.6}$$

where $\boldsymbol{\delta}^{l+1}$ is already known from the $(l+1)^{\text{th}}$ (in this specific example, L^{th}) layer, as we are propagating back through the network. $(\boldsymbol{W}^{l+1})^T$ is the transposed weight matrix for the $(l+1)^{\text{th}}$ layer. We then apply the transposed weights to $\boldsymbol{\delta}^{l+1}$. After taking the Hadamard product, we are left with the vector of errors $\boldsymbol{\delta}^l$ for layer l. This means that if we combine the two first equations of back propagation, equations 4.5 and 4.6, we are able to compute all error vectors of the network, starting at the final output layer L (Nielsen,

[5]Hadamard product is simply a way of multiplying each index in a matrix $\boldsymbol{A}$ with the corresponding index in another matrix $\boldsymbol{B}$.

2015).

Finally, we are now able to compute the two goals we introduced in the beginning of this section. First, we present the third equation of back propagation; the rate of change of the cost, with respect to any bias:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{4.7}$$

We already know how to compute δ_j^l from equations 4.5 and 4.6. This means that the error δ_j^l is equal to the partial derivative of C with respect to the bias at any node j in any layer l. The fourth and final back propagation equation presents the rate of change of the cost with respect to any weight:

$$\frac{\partial C}{\partial w_{j,k}^l} = a_k^{l-1}\delta_j^l. \tag{4.8}$$

We already know δ_j^l and a_k^{l-1}, which made it possible find the partial derivatives $\partial C/\partial w_{j,k}^l$.

By applying equations 4.5, 4.6, 4.7 and 4.8, we are able to back propagate through the entire network, update weights and biases and thereby reduce the cost of the network (Nielsen, 2015).

4.2.4 Nonlinear Problems

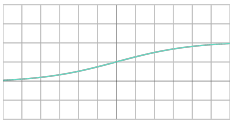
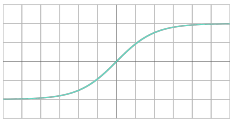
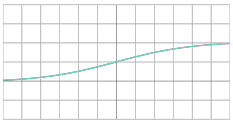
In a neural network, activation functions are the functions that define a node's output. Considering a biological neural network, we consider a neuron's activation the rate of action potential firing (Hodgkin and Huxley, 1952); activations are considered the amount of potential effect nodes have.

We previously discussed predicting some output by finding the optimal weights and biases using back propagation. Equation 4.1 is a linear function — a function producing linear output. Given this linear combination, we are only able to estimate data that is linear separable. However, data tends to be complex and often easier to classify with nonlinear methods. To handle such data, we can apply a nonlinear function to the weighted input z_j^l that produces an activation output a_j^l. Neural networks with applied activation functions tend to be a good choice for nonlinear classification problems (Goodfellow et al., 2016).

4.2.4.1 Applied Activation Functions

This thesis focuses on three activation functions, namely the sigmoid, TanH and softmax functions. By squashing some input[6] x, these functions produce some output[7]. Table 4.1 lists activation functions applied throughout this research project (Goodfellow et al., 2016).

Table 4.1: Applied Activation Functions

Name	**Equation**	**Range**	**Graph**
Sigmoid (Logistic)	$\sigma(x) = \frac{1}{1+e^{-1}}$	$(0, 1)$	
TanH	$\sigma(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$(-1, 0)$	
Softmax	$\sigma(x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1, ..., K.$	$(0, 1)$	

All equations listed in table 4.1 appear nonlinear, allowing neural networks to train on data of higher complexity.

Sigmoid One widely-applied activation function is the sigmoid function. It simply takes the input and squashes it into a value of conditional probability between 0 and 1. This way, we can use the sigmoid function to map the input to a class such that

$$y = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}.$$

Sigmoid is particularly suited to binary classification problems, where the outcome is one of two possibilities.

TanH TanH is a rescaled version of the sigmoid function. Output ranges from -1 to 1 instead of 0 to 1. Highly negative inputs will also be negative outcomes. This property makes the TanH less likely to output activations around zero. TanH is applied in cells of recurrent neural networks, as covered in depth in section 4.3.

[6] typically the weighted input previously defined as z_j^l.
[7] previously defined as a_j^l.

Softmax Another useful activation function is the softmax function. It is basically a generalization of the sigmoid function. Much like the simpler sigmoid function, it squashes a node's input vector, and returns the equivalent probabilities summarized to 1. However, softmax can work with problems of multiple classification, whereas sigmoid is focused on binary classification problems.

4.3 Recurrent Neural Networks

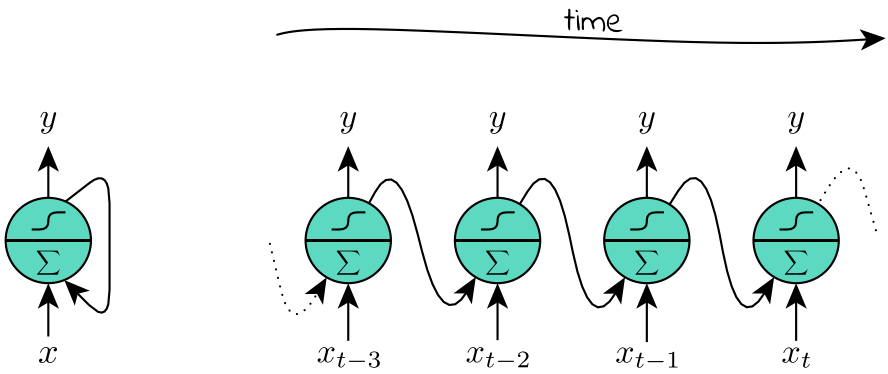
In this section we present a subclass of artificial neural networks called *recurrent neural networks*. Recurrent neural networks act much like the feed forward network, except for having connections pointing backward through time. Recurrent neural networks are a known tool for analyzing time series data and text. Recently, this class of network has achieved remarkable results in areas such as natural language processing and speech recognition. These types of data have something in common, namely *sequences* of data observations. Recurrent networks can work with sequences of arbitrary lengths rather than sequences of fixed lengths. This, for example, allows for the analysis of sentences of variable lengths, making them effective for tasks such as translation and sentiment analyses (Goodfellow et al., 2016; Gron, 2017; van den Oord et al., 2016; Yogatama et al., 2016).

Lately, Google has been using recurrent neural networks to create melodies. By training the recurrent network on melodies, the network is able to predict the next tone. For example, by feeding a trained recurrent neural network model a tone, one can make the network predict the tone most likely to follow in a melody. This following tone can be re-fed to the network and so on, making the network generate a full-length melody (Google-Brain, 2016). The same concept holds when predicting sentences using a recurrent neural network.

4.3.1 Basic Recurrent Elements

The *recurrent* part of the network is in the nodes. A recurrent node produces some scalar output y, based on some scalar input x, as well as its own output from previous *timestep*, x_{t-1}. To illustrate the time dimension, figure 4.4 shows a folded recurrent node on the left and the version unfolded through time on the right. As mentioned in section 4.2.2, each node has a weighted input and an activation output. The weighted input is symbolized using a sum, $\sum$. The activation is simply illustrated using a s-curve. Note that the activation output of the node in time $t-1$ is fed as input to the same node in time t — the recurrent element of these networks.

For simplicity, figure 4.4 illustrates a single recurrent node. In reality, nodes are often presented in layers of nodes. The concept of a layer of nodes is the same as the single



Source: Own elaboration.
Figure 4.4: Recurrent node (left), unfolded node through time (right).

node presented in figure 4.4, except that a layer of nodes accepts and outputs a vector of values rather than a scalar. Much like equation 4.2 calculates the activation output, the output of a single recurrent layer, $\boldsymbol{Y}_t$, can be written as:

$$\boldsymbol{Y}_t = \sigma(\boldsymbol{X}_t \odot \boldsymbol{W}_{\boldsymbol{X}} + \boldsymbol{Y}_{t-1} \odot \boldsymbol{W}_{\boldsymbol{Y}} + \boldsymbol{b}). \tag{4.9}$$

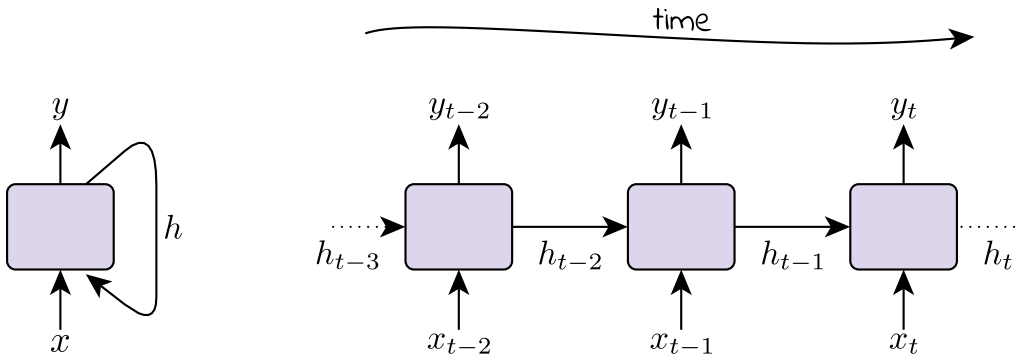
The main problem recurrent neural networks face are vanishing or exploding gradients. The issue was explored in depth by Hochreiter (1991) and Bengio et al. (1994). This project will not explore the technicality behind vanishing gradients in depth, but will briefly discuss the problem as follows. Vanishing gradients basically means gradients shrinking to zero, making nodes saturate. This means the network is unable to learn long-term dependencies. This is particularly an issue in deep neural networks. Recurrent neural networks easily become deep as the number of timesteps increases, making the issue of vanishing gradients current for this class of networks. Vanishing gradients eventually lead to the development of long short-term memory and gated recurrent unit architectures, as presented later in section 4.3.2.1 and 4.3.2.2.

In order to work with recurrent neural networks, one must choose the number of timesteps. For example, if a network is to predict the next word of a sentence (or the next tone of a melody), the network will need to know about the previous words of the sentence. The number of words, in this case, would be the number of timesteps back in time. For each timestep forward in time, the network will take into account the previous output computed upon previous words, consequently making the network *remember*. In short, the output of previous timesteps affects future timesteps; we refer to this as memory of a network. Gradient contributions from "far away" timesteps may eventually turn to zero, making the network unable to learn (vanishing gradients). The following section briefly covers the theory behind the memory of recurrent neural networks.

4.3.2 Complex Cell Architectures

In section 4.3, we presented equation 4.9, where the output, $\boldsymbol{Y}_t$, is a function of $\boldsymbol{X}_t$ and $\boldsymbol{Y}_{t-1}$, which is a function of $\boldsymbol{X}_{t-1}$ and $\boldsymbol{Y}_{t-2}$, and so on. Outputs of recurrent nodes and layers depends on previous timesteps, hence recurrent networks are built on memory. The part of a neural network that preserves states across time is called a memory cell. This means that the recurrent node and layer presented in previous section 4.3.1 are referred to as a basic memory cell (or simply cell). Figure 4.5 illustrates a single recurrent cell on the left and the unfolded version on the right. As explained in the following sections, the structure of cells is much more complex than that of a single node.

Figure 4.5 shows some output h_t; the hidden state of the cell. In the case of the basic cell presented earlier, the output y_t, of the cell, is equal to that of the hidden state, h_t. However, cells with more complex architectures may have an output y_t that is not necessarily equal to the hidden state, h_t (Gron, 2017; Olah, 2015).



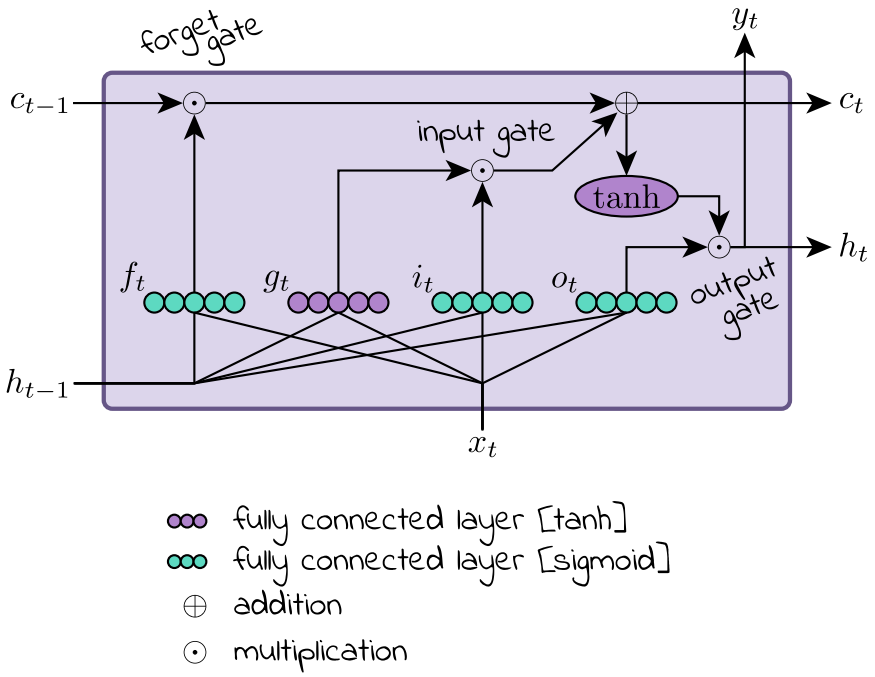
Source: Own elaboration.
Figure 4.5: Memory Cells (left), unfolded cell through time (right).

The following sections briefly cover the theory behind the long short-term memory and the gated recurrent unit cell architectures.

4.3.2.1 Long Short-Term Memory

The Long Short-Term Memory (LSTM) cell was developed by Hochreiter and Schmidhuber (1997) and has since been improved by other researchers. LSTM, like the basic cell, computes a new state and an output from an input and the previous state. The (hidden) state of LSTM is split in two vectors, one for short-term state and one for long-term state. Figure 4.6 illustrates the architecture of the LSTM cell. In figure 4.6, c is the long-term state and h is the short-term state. LSTM receives both short-term and long-term states from previous timesteps (Gron, 2017).

Instead of a single node layer, the LSTM cell holds four fully connected layers. The main layer, g_t, is comparable with the layer of the basic cell presented earlier. Isolating



Source: Own elaboration. Inspired by Gron (2017).
Figure 4.6: Long Short-Term Memory Cell.

this layer without any other operations would result in a basic cell with a single recurrent layer. In LSTM cells, the output of g_t does not flow straight out as in a basic cell. Instead, g_t is passed on for partial storage in the long-term state. The other three layers are called gate keepers, as their output toggles the gates in the cell. They all use the sigmoid activation function, resulting in outputs between 0 and 1, as covered in section 4.2.4. Specifically, if they output zeros, the gates will close and but otherwise open. According to Gron (2017), gates can be understood as follows:

Forget gate is controlled by f_t. It controls which parts of the long-term state, c, to take out. The role of this gate is to forget input that is no longer important.

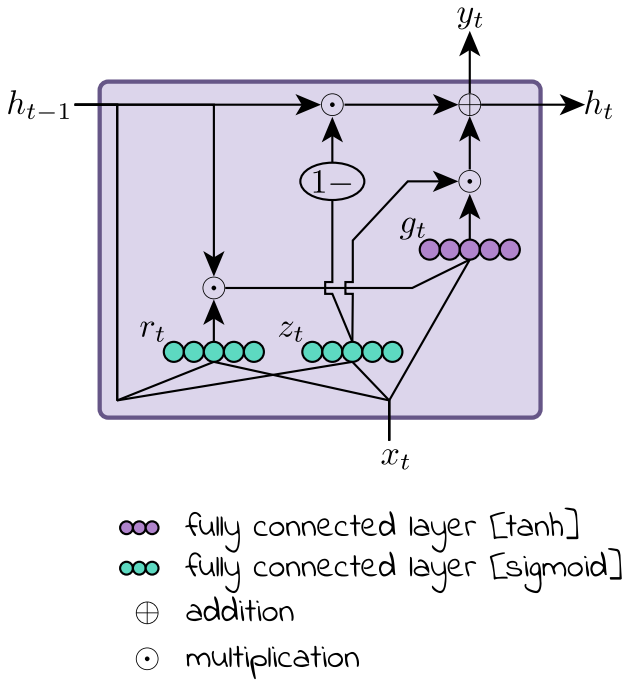
Input gate is controlled by i_t. It controls which parts of the main layer, g_t, to add to the long-term state. The role of this gate is to recognize an important input and store this in the long-term memory to preserve it.

Output gate is controlled by o_j. It controls which parts of the long-term state to output at current timestep, t.

The following section cover a particular branch of LSTM called The Gated Recurrent Unit (GRU).

4.3.2.2 Gated Recurrent Units

This architecture was first presented by Cho et al. (2014). GRU is a less complex architecture, but still seems to perform as well as LSTM (Chung et al., 2014). Figure 4.7 illustrates the GRU cell. In GRU, both state vectors of the LSTM have been merged into a single vector, h. Instead of three gate controllers, GRU uses two; one controlling both the forget gate and the input gate, while there is no output gate. The full state vector is the output at every timestep.



Source: Own elaboration. Inspired by Gron (2017).
Figure 4.7: Gated Recurrent Unit Cell.

LSTM and GRU tend to have almost equal results, but GRU is less complex and often trains faster than the LSTM version. The choice of whether to utilize LSTM or GRU may depend on the data (Chung et al., 2014).

4.4 Summary

This chapter presents a new technique for modeling complex data with the purpose of predicting some outcome based on the input. This technique is based on artificial neural networks — layers of connected nodes. First, we present a basic version of artificial neural networks. Training these networks happens by feeding the network training data in batches. In order to determine which weights to adjust, a cost (or, loss) of the network is calculated once predictions are made. The cost is a measure of distance between the output predictions and the actual training targets. The lower a cost gets, the better

predictions the network produces. The cost is used to determine which weights and biases to update using an optimization algorithm. Optimization happens using backward propagation; an advanced algorithm for updating weights and biases backward through the network. One forward and one backward propagation makes a training iteration.

Recurrent neural networks, a subtype of artificial neural networks, are presented. This technique specializes in sequential data in order to predict forthcoming elements of sequences. This type of network includes recurrent cells; cells that keep memory states. We present LSTM and GRU, two popular cell architectures. These cells use their own outputs as inputs. This way, previous timesteps of sequences affect future timesteps. This will be useful when working with sequential data such as customer sessions.

CHAPTER 5

Evaluation Methods

Any accurate recommender system plays an important role for e-commerce businesses, as these systems engages the customers and provide relevant information supporting customers' decision making and searches. Overall, recommender systems can lead to increased customer satisfaction, revenue and brand value (Ricci et al., 2011). Inaccurate or non-existent recommender systems that fail to support customers decisions' may result in lost customer time as customers are forced to wade though irrelevant products. This frustrating process increases the risk of losing customers and ultimately results in a lower conversion rate (Kunaver and Pozrl, 2017). A quick and accurate recommender system strives to ease the customer process of finding a desirable product. Specific criteria and metrics ensure that the proposed models make accurate recommendations.

5.1 Settings

Online and offline evaluations have both strengths and weaknesses. In an online setting, user reactions towards recommendations are analyzed. Online evaluations can discover the underlying reasons for users' trust and confidence in the system (Herlocker et al., 2004). Online evaluations, such as surveys, controlled lab simulations and live A/B testing done by Herlocker et al. (2004) reveal important factors. Herlocker et al. (2004) found that particularly trust and confidence play a significance role in the users' satisfaction with recommendations.

During this research project, online testing is not an option, because online testing requires implementations of developed recommendation systems. Furthermore, such data would require a significant period of time to collect. To compensate for the lack of online testing, multiple metrics will determine the model's performance. When comparing performance of models with performance of the baseline model, a common metric continues to be accuracy, but accuracy alone should not evaluate the performance (Herlocker et al., 2004).

Usually, when working with recommender systems, offline evaluation methods are

applied. Offline testing takes advantage of historical data and splits all data into a training set and a testing set. When modeling data, the training part happens on the training set and testing of performance occurs on the testing set. Only predictions made using the testing set are relevant as evaluation metrics, as predictions made in the training set are subjects of bias and may suffer from overfitting (Herlocker et al., 2004).

5.2 Criteria

A recommender system has four primary goals to ensure helpful recommendations. These goals can be difficult or impossible to measure when doing offline testing (Herlocker et al., 2004). Measuring the relevance of the recommendations will remain the most important of the goals. Without relevance, a novel or diverse recommendation will neither support nor interest the user.

In order to develop a trustworthy — and customer decision beneficial — model, it is crucial to evaluate the model correctly. Wrong measurement will falsely determine the goodness of a model. The proposed models should seek to outperform the baseline model on these parameters and thereby seek to show the effective usage of recent research in the area of deep learning for session-based recommendations. Four different criteria will evaluate the final models.

Accuracy Accuracy measures the relevance of predictions. We assume that the next item of a user sequence is relevant if a user actively chooses to view this item. We decided to use the same accuracy metric as Hidasi et al. (2016) in order to compare results. Specifically, accuracy is measured by checking whether the target item is among the top 20 predicted items. If the target item is among the top 20 predicted items, the accuracy score increases. The total proportion of times that the target item is among the 20 predicted items determines the final accuracy.

This accuracy rate lays the groundwork for a metric that compares the model's performance against the baseline model and will work as the most important metric. At evaluation times, a vector $\boldsymbol{p}$ should hold all predictions; ones when target was found among predictions and zero if not. Equation 5.1 shows the calculation of the total accuracy, acc, given a vector of true/false predictions, $\boldsymbol{p}$:

$$acc = \frac{\sum_{i=1}^{n} \boldsymbol{p}_i}{n}, \tag{5.1}$$

where n is the total number of predictions and $\boldsymbol{p}$ is a vector of length n holding true/false predictions. By dividing the predictions by the total number of predictions made, we determine the total accuracy.

Novelty Repetitive predictions in a session will be difficult to measure as this is computationally expensive and numerous of the sessions length will have limited events. Post-processing methods can ensure novel recommendations.

Diversity As discussed earlier, diversity can be difficult to derive. Furthermore, the required computation power required if accounting for diversity is significant. Instead, we will suggest a post-model method for filtering out items of less diversity.

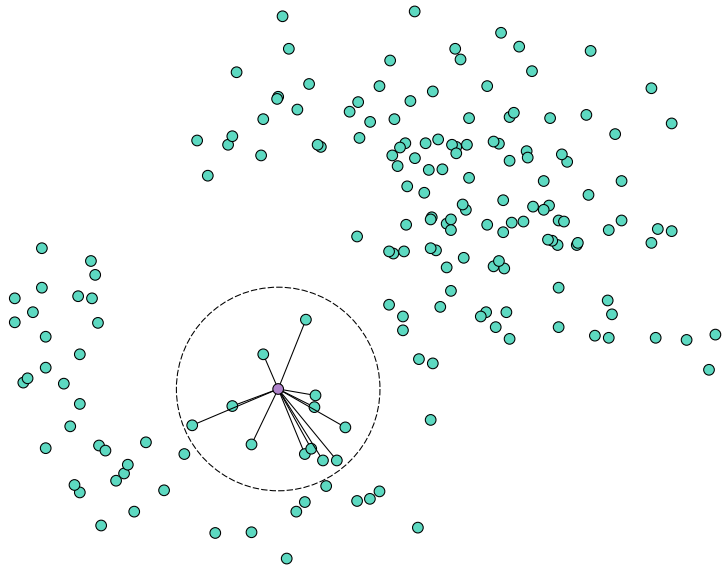
Serendipity This metric is impossible to measure when doing offline testing, as the users cannot give answers to the serendipity of their recommendations. However, this can be theoretically explained and accounted for by improving the novelty and diversity.

All criteria will be essential when deciding on best performance models. Furthermore, as briefly discussed in section 3.1.2, we present a baseline model that will be used for benchmarking and to determine if the applied deep learning techniques are better at mining frequent patterns in customer behavioral data.

5.3 Benchmarking Baseline

This research focuses on models built with artificial neural networks, but we find it important to include a baseline model to benchmark results. The baseline model will briefly be covered in the following.

The algorithm used for a baseline model is k-nearest neighbor. According to Hidasi et al. (2016), kNN outperforms other models outside the scope of deep learning. Hidasi et al. (2016) tested on the RecSys 2015 dataset — see section 6.2.1 for details on this



Source: Own elaboration, inspired by Bernhardsson (2015).
Figure 5.1: Two-dimensional representation of the kNN baseline model.

dataset. Usually kNN is based on user ratings as inputs. This research uses user-visited items as ratings. Hence an item rating for one user equals one if this user visited the item and zero if this user did not visit the item. As discussed in section 3.1.2.1, the cosine distance function will compute the similarities between items based on sessions.

Items often visited together will tend to be closer in distance and will ultimately form groups of neighbors. Figure 5.1 shows how items form groups in the two-dimensional space. The figure is a basic illustration of kNN determining which observations are neighbors.

5.4 Summary

Evaluation is a large part of determining the best models. First, we discuss online and offline evaluation methods. Online is not possible due to limited research time. Offline measures are used to benchmark models. Accuracy is the main criteria for model performance but several other elements help determine good models; novelty, diversity, serendipity. This chapter also presents a brief section of the baseline model; a model used to benchmark results produced using deep learning techniques.

Part III

Methodology

CHAPTER 6

Data

"Without big data analytics, companies are blind and deaf, wandering out onto the web like deer on a freeway."

Geoffrey Moore, Tweet 12th Aug. 2012

The quality of data has significant effect on the performance of a model (Kan, 2003). Analyzing and transforming the data plays an important role to ensure quality output and in this case to ensure valuable recommendations, making customers' decisions easier. E-commerce webstores may generate large amounts of data, but only some of this data is useful when recommending items to new and previously unseen customers (Bidgoli, 2003).

Some recommender systems currently require significant amounts of data about users to make recommendations. Often, it is necessary for users to register, log in and explicitly or implicitly rate items before they receive good recommendations (Hidasi et al., 2016). Users dislike this high level of engagement and are easily dissatisfied if no items presented are interesting for them (Kunaver and Pozrl, 2017). A large number of websites have switched to item-based collaborative filtering. These simple item-based collaborative filtering models provide useful results, but they merely consider recently viewed items of the user and have drawbacks, as discussed in section 3.1.2.1.

Like most modeling techniques, deep learning techniques require some data preprocessing. A large part of this research was spent collecting data from databases, restructuring and transforming the data. All data preprocessing has been written into a single script, allowing for quick generalization of the preprocessing when new data is collected.

6.1 Type

The widely-used collaborative filtering recommender systems use user data, such as age, gender, location and previously purchased, rated or viewed items (Ekstrand et al., 2010). Most of this data requires a period of time to collect and users are typically required

to create a profile (log in) in order for the site to collect such data. To make quick predictions based on the user behavior, the model needs instant data collection without explicit engagement from users. Therefore, the proposed models will only use the click-stream session data, as this requires no explicit user engagement and collection happens instantly.

A session is defined as the period in which a user is active on a website. Activity can be almost any user engaged action, such as viewing items. A session ends once a user is no longer active. This research defines inactivity as any period with 30 or more minutes between item views, hence sessions will be cut once a period of more than 30 minutes elapses between a users' item views. However, a session's length can vary depending on the chosen time of inactivity required for a session to end. The data utilized in this research contains sequences of item views on user level.

We define an event as a user visiting an item. Each event must hold the below information of user behavior, as these are crucial to ensure quality and ultimately extract the user's intent:

1. a unique session ID (`session_id`).
2. a unique item ID of the visited item (`item_id`).
3. a timestamp of the event to identify the sequence (`timestamp`).

This data is generally logged on any e-commerce webstore. To ensure that the resulting model is robust and generalizable, only the above session data is assessed during this research.[1]

6.2 Session Data Sources

A dataset holds all the information needed as covered in the previous section. A single dataset holds this information from a single e-commerce website. This research is based on two datasets from e-commerce websites. To support the proposed model, the use of two datasets will help in validating the performance[2]. By applying the proposed models on two datasets with different sizes of item catalogs, we can challenge the performances dealing with various numbers of unique items.

The datasets used consist of session data collected from the RecSys Challenge 2015 and the e-commerce webstore Audio Visionary Music (imusic.dk). The following sections

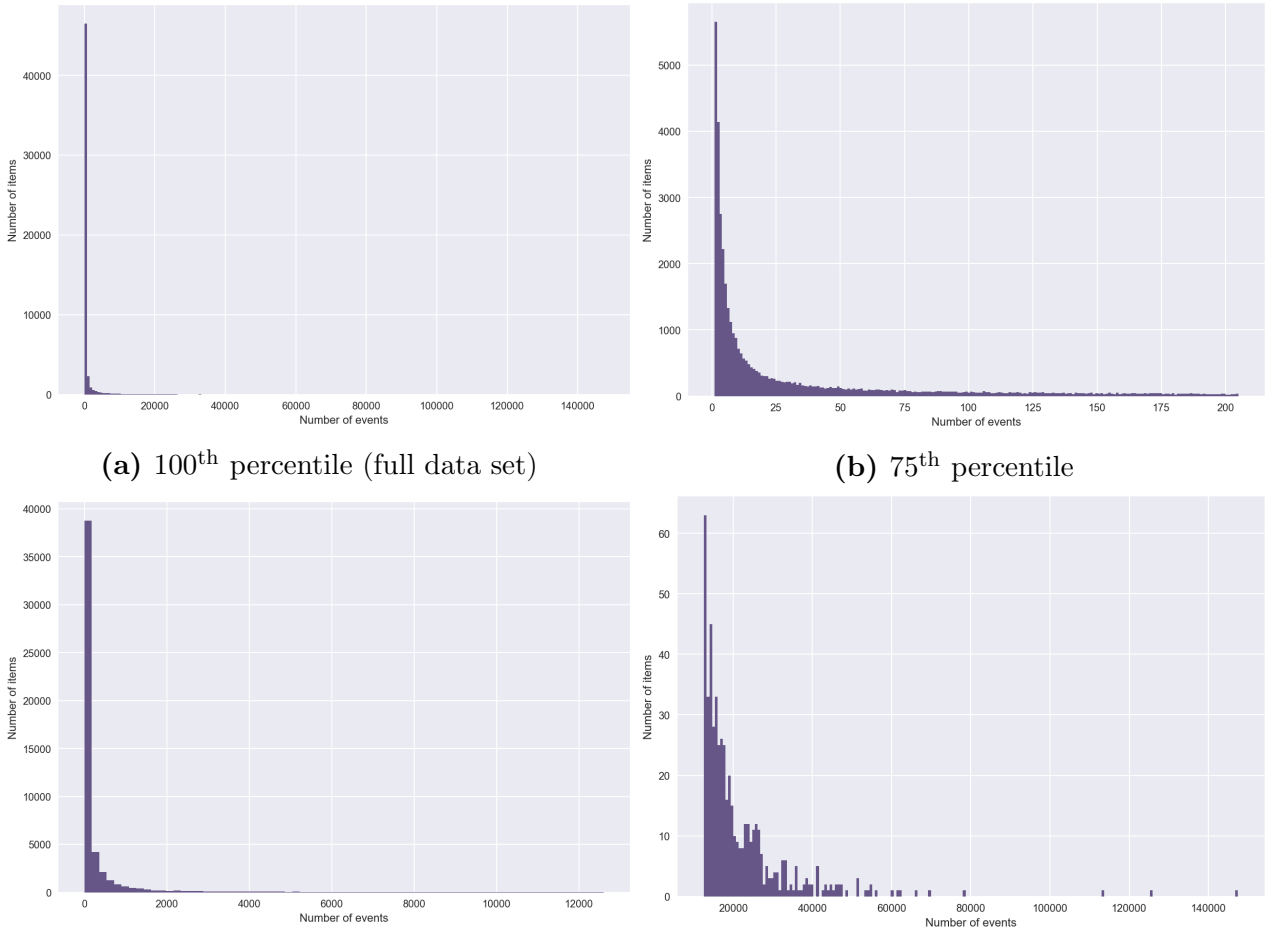
[1]It is arguable that multiple data sources may lead to a better model. In this research, we decided to aim for a broad recommendation engine, rather than an engine using multiple data sources only available on one or few e-commerce websites.

[2]Note that in section 7.2.2 we discuss the need for adjustments of the hyperparameters between datasets when training the models

analyze the data by illustrating frequency plots.[3]

6.2.1 RecSys Challenge 2015

This dataset was part of the RecSys Challenge 2015 (we call it RecSys), where contesters had to predict correct recommendations based on a sequence of click events. The dataset contains 33,003,944 events: 9,249,729 sessions and 52,739 unique items. All clicks were registered on a European e-commerce site over a period of six months and the product range consists of toys, clothes, electronics and more. Other specificities of the data source are omitted (RecSys, 2015). The RecSys dataset has already been an asset in areas of research (Hidasi et al., 2016).



(a) 100^{th} percentile (full data set)

(b) 75^{th} percentile

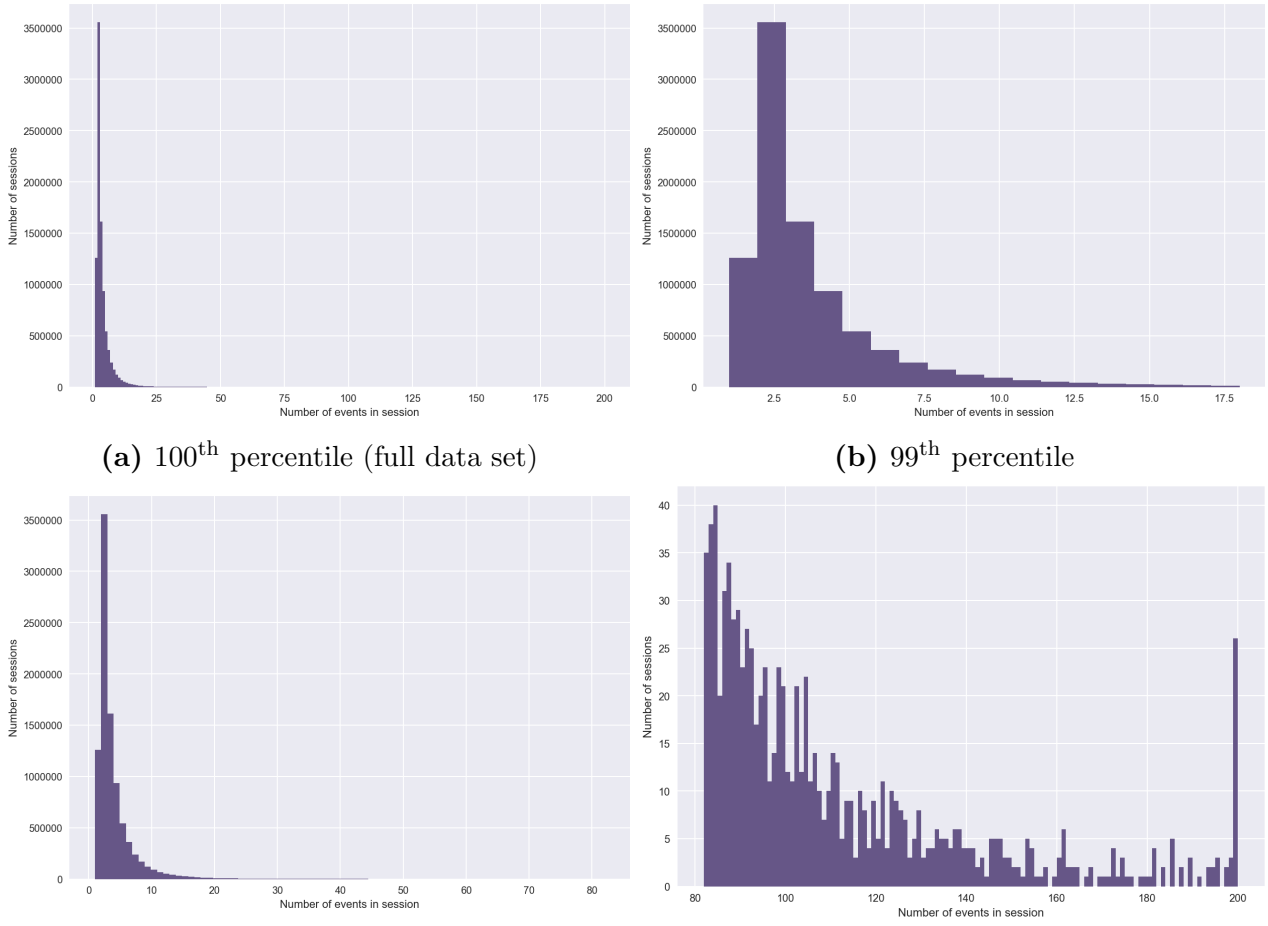
(c) 99^{th} percentile

(d) 99^{th} to 100^{th} percentile

Source: Own elaboration.
Figure 6.1: RecSys: Distribution of item views.

Figure 6.1 shows the distribution of item views in the RecSys dataset. Specifically, the figure shows highly right-skewed distributions between the number of items and number of

[3]Plotting of data is done in batches (bins) due to the immense amount of data.



(a) 100^{th} percentile (full data set)

(b) 99^{th} percentile

(c) 99.99^{th} percentile

(d) 99.99^{th} to 100^{th} percentile

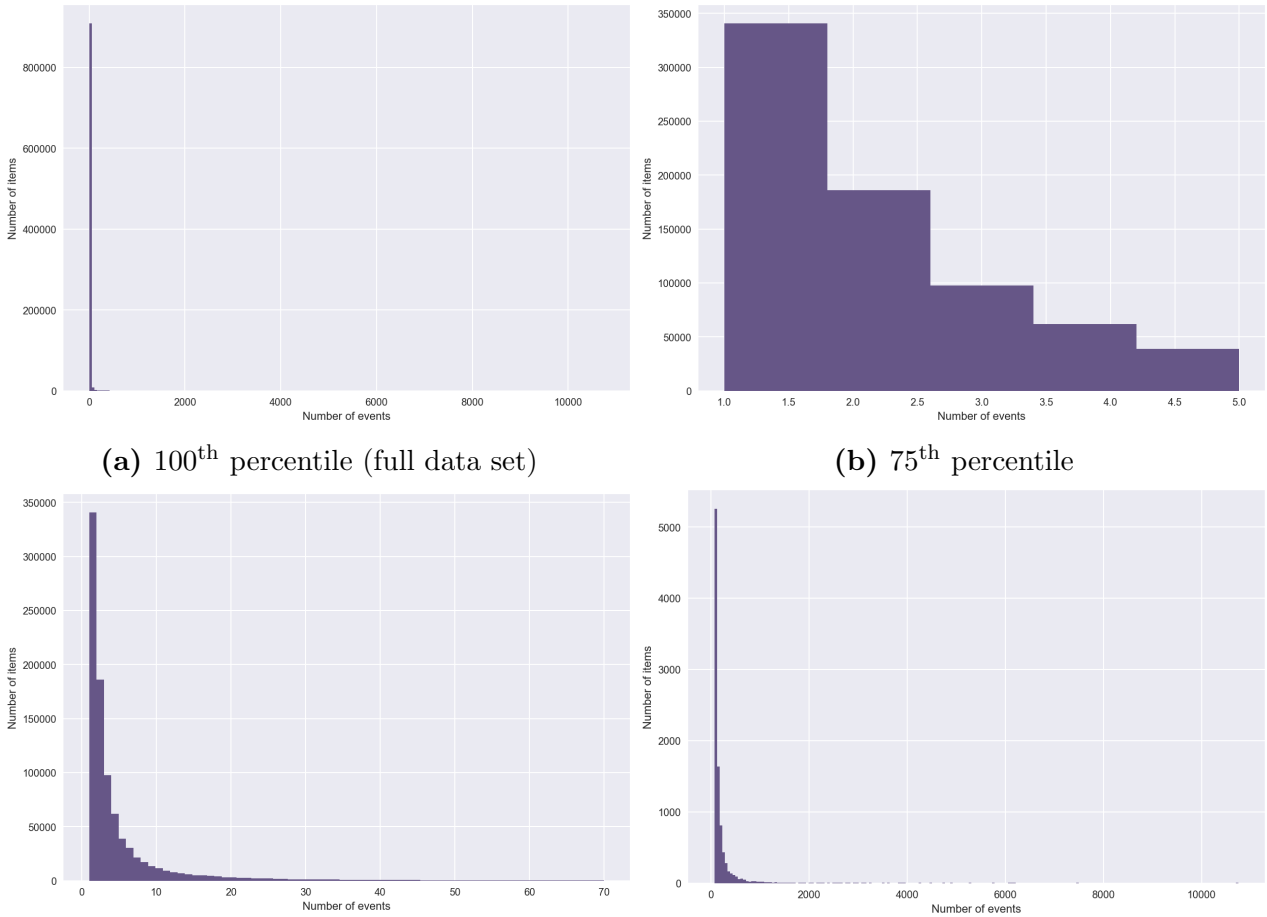
Source: Own elaboration.
Figure 6.2: RecSys: Distribution of sessions.

item views. A high number of items have a very limited number of views. As discussed in chapter 3.3, this is a signal of sparse data, which may ultimately affect the performance of a model negatively. To mitigate the risk of poor performance on sparse item data, we can remove items from the dataset that were visited fewer than n number of times. Likewise, we see that nearly all items have less than ≈ 25 views, particularly illustrated in figure 6.1b.

To get an idea of session lengths, figure 6.2 is illustrated. We see that a high number of shorter sessions carries most of the dataset. In particular, we see that sessions of one event (i.e. one item view per user session) counts $\approx$ 1,250,000 and sessions with two events counts $\approx$ 3,500,000, indicating that short sessions dominate the data. Figure 6.2d provides insight into sessions expanding above 80 events. This span of events can become an issue when modeling the data, as recurrent neural networks operate with fixed sized inputs (Mao et al., 2014). Longer sessions will need some data transformation, as discussed in section 6.3 of this chapter.

6.2.2 Audio Visionary Music

Audio Visionary Music (we call it AVM) has grown rapidly over the last 10 years. They serve around 70,000 orders on a yearly basis, selling music, movies, games and books. AVM has a total product repertoire of more than 15 million products, resulting in complex product data due to more than 1,000 different suppliers in total. The complexity of product information expands as almost no supplier delivers the same types of information. As an example, one supplier may categorize Bruce Springsteen's first album as rock, while another supplier may categorize his next album as soft-rock or perhaps ambient-rock. These twists complicate content-based filtering, since items hold very different data. Instead, we seek to apply the session data of AVM on the collaborative filtering method in question, looking only at the session traces of item views by users. Currently, AVM has no recommendation engine. However, it manually promotes top 40s and other lists of products. Based on the scale of products available, AVM is the largest media e-commerce site in Scandinavia.



(a) 100^{th} percentile (full data set)

(b) 75^{th} percentile

(c) 99^{th} percentile

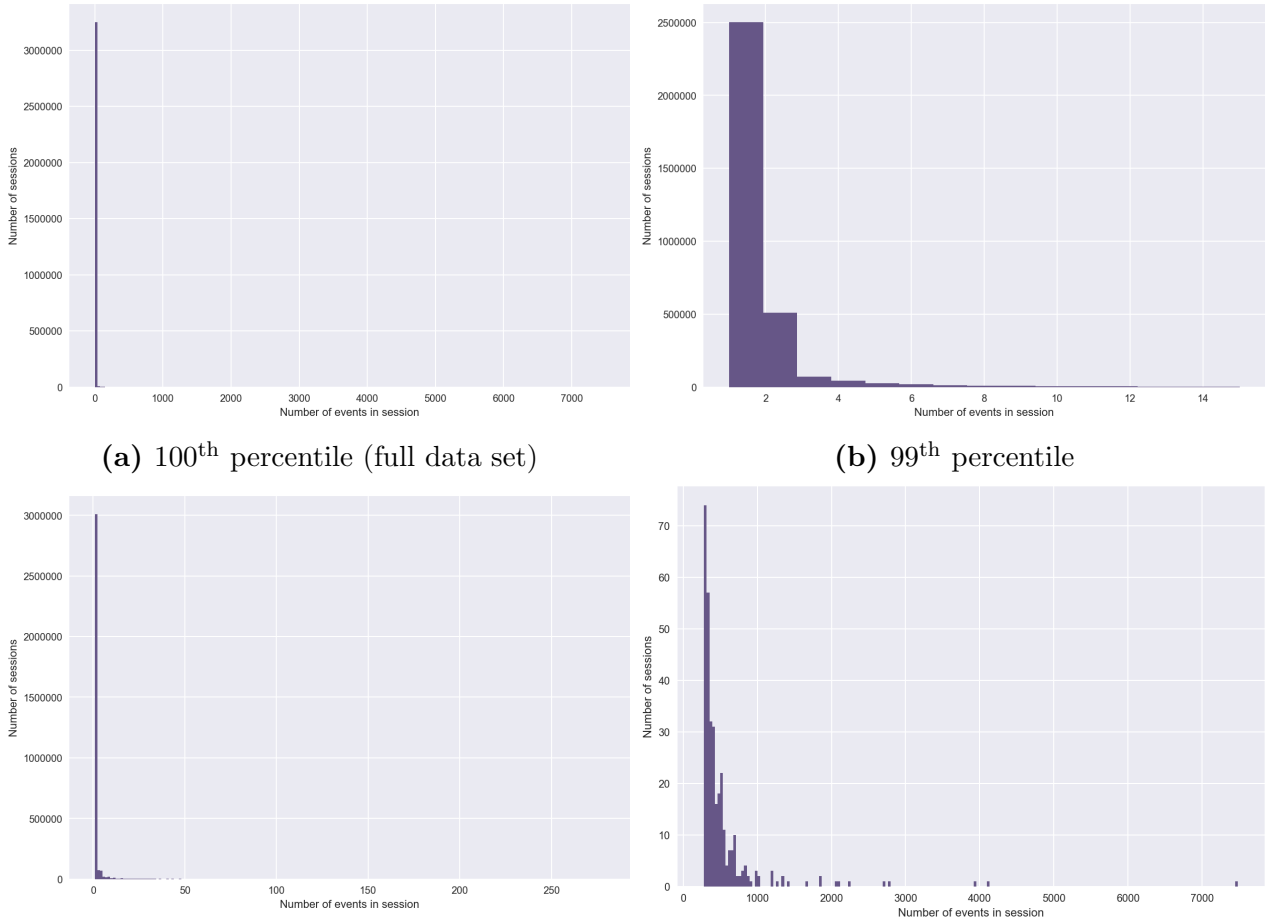
(d) 99^{th} to 100^{th} percentile

Source: Own elaboration.

Figure 6.3: AVM: Distribution of item views.

Figure 6.3 shows item views and the number of items in each bin. The largest portion of items have under 10 item views. The large proportion of less-viewed items is not surprising as 922,271 different items have been viewed in this dataset. A much smaller portion of the items has a higher number of views. 99% of the items have under 70 views, which may become a challenge when training the models for AVM. The massive number of products available at AVM leads to few events on almost all products, forcing us to reduce the data. This is done by removing items with five or fewer events; 75% of this dataset consists of items with five or fewer events. The manually crafted static lists of top-40 products at AVM seems to make a few products highly popular; some products have $\approx$ 10,000 views.

The dataset consists of 3,259,300 sessions and 6,050,457 events as illustrated in figure 6.4. The distribution is right-skewed, as observations are concentrated near to zero. Some sessions lengths are over 500 and some close to 7,000, which may indicate robotic events. Furthermore, about 2,500,000 sessions have merely a single event.



(a) 100$^{\text{th}}$ percentile (full data set)

(b) 99$^{\text{th}}$ percentile

(c) 99.99$^{\text{th}}$ percentile

(d) 99.99$^{\text{th}}$ to 100$^{\text{th}}$ percentile

Source: Own elaboration.

Figure 6.4: AVM: Distribution of sessions.

6.3 Cleaning

This section dives into the data preparation and cleaning methods applied to both datasets. Depending on the data source, the applied methods may vary. Subjects of this section need attention to ensure the quality and format of the data.

Robots For large e-commerce sites, robots are almost inevitable. Shilling attacks can affect some recommender systems. The AVM dataset contains robots mainly indexing, but also scraping or shilling the e-commerce site. Since we use events as ratings, robots generate noise in the data. These robots act as users visiting a site, consequently requiring a method of identifying robots in order to ignore robotic sessions during the training of the models. Robots were effectively removed by taking out sessions meeting both of the following requirements:

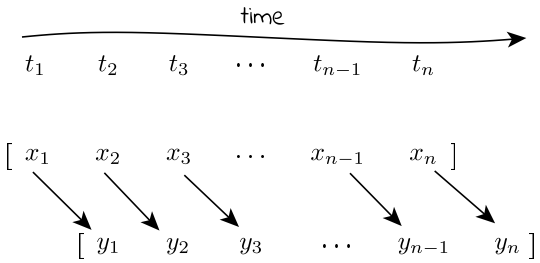
- Sessions having more than five item views.
- Total spent seconds of a session is less than the total amount of events, i.e. less than one second spent per event.

This way, sessions with more than five events and less than a second between each event indicates the user is neither human nor interested in the items visited.

Rare Items When looking at figures 6.1 and 6.3 it seems obvious that both datasets contain a high number of rarely-visited items. These sparse events tend to be noise, contributing less when training the final model and they may eventually decrease the final accuracy. By removing items with lower than five views, the data is left with items that tend to be interesting for users which may ultimately empower the final model by increasing accuracy. However, removing rare items makes the diversity — the prediction of the long tail data — worse (Hearty, 2016). During this research project, we choose to prioritize accuracy rather than diversity, simply due to the fact that both datasets contain immense item numbers and so full diversity will not be possible. As a goal of this research remains to be providing new customers with accurate recommendations, in order to grab customers attention, accuracy is of most interest.

Session Length The dataset structure contains rows of events made of three columns: `session_id`, `item_id` and `timestamp`. In each session, the number of events is also referred to as the number of *timesteps* in a session. Sessions with a single timestep (one event) are dropped as it is not possible to train a model on inputs with no targets[4]. The remaining sessions will be split into input and target values.

[4]Sessions of a single event do not provide information about which item comes next.



Source: Own elaboration.
Figure 6.5: Splitting sessions into input and target values.

Figure 6.5 was made to illustrate the splitting of a session into x inputs and y targets over timesteps, t.

The span of session lengths can be a problem for recurrent neural networks as they require fixed sized inputs. To assess this issue, all shorter sessions will be padded n times, where n is the number of timesteps in the longest session. However, by applying this padding technique, the size of input data increases by a factor of 40 for the RecSys dataset while the AVM input data increases even more. Rather, we argue that a smaller number of timesteps still represents the intent of the users. The chosen number of timesteps determines how far back in time the model will look.

The main proportion of the RecSys and AVM sessions spans 19 or fewer events. As the model seeks to understand and recommend items to new users, a timestep of 19 for both datasets seems a good choice. This number of timesteps increases the input data size by less than a factor of four — a much more feasible factor. Sessions with more than 19 timesteps are split into multiple sessions and act as separate independent sessions. Loss of information by splitting long sessions is substantially low and the advantages of a much higher computation speed is valuable.

Padding A method called *zero padding* makes sessions the same length by adding zeros to missing timesteps in sessions shorter than n (Hearty, 2016). Padding is later reversed by masking, which ensures the added zeros of padding have no effect on model performance.

Cleaning both datasets reduced the size of the datasets, in particular the AVM dataset. Cleaning the RecSys dataset of 37,483 items leaves 7,981,581 sessions and 31,708,461 events; 15,256 items and 1,268,148 sessions were removed. This reduction was mainly due to the high number of $\approx$ 1,250,000 sessions with one event. The reduction in the AVM dataset was much higher due to a high bounce rate[5] and thereby a lot of sessions without usable information. Cleaning of the AVM dataset left AVM data with 138,579 items, 595,653 sessions and 2,571,605 events; 783,692 items and 2,663,647 sessions were

[5]The proportion of users entering the webstore and leaving before performing a second event.

removed. As we saw in figure 6.4b, AVM had ≈ 2,500,000 sessions with only one event, indicating that AVM suffers from a bounce rate of around 75%. In addition to this, the AVM data suffers from sparsity, with an *events per item* ratio of 18.56 compared to RecSys's 845.94. To ensure the best model, only data of quality is trained upon; reducing the dataset sizes was essential.

6.4 Partitioning

Validating of the models must happen on a test set with holdout data[6]. The training set will not be used to evaluate the final model, as the weights and biases of the model are tuned to predict targets of the training set. Hence, an unknown set of data must validate the final model, leaving an indication of how well the model performs on new (unseen) data.

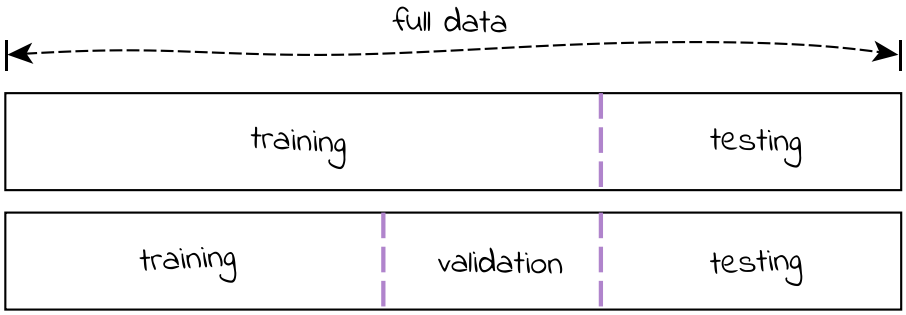
In general, two methods can help in testing this: k-fold cross validation or a two/three-split of the data. The k-fold cross validation divides the dataset into k subsets. Training happens iteratively on k - 1 subsets while testing on the last subset. Testing this way is computationally expensive since training/testing happens k times and finally averages the accuracy. The sizes of the datasets applied in this research make k-fold cross validation a long-term task.

We furthermore expect some degree of temporal changes in users' preferences, which cross validation does not account for. RecSys data was gathered over a period of a minimum of 6 months in which we expect users' preferences to change considerably as trends arise and seasons affect the popularity of items. However, the two/three-split method can account for these changes in time. Splitting the dataset according to normal practice means randomly selecting observations. We propose that this selection must not be random due to temporal changes. Instead, the split was deliberately done on the last periods of the datasets. This ensures the model cannot train on observations after the test set.

Specifically, Hidasi et al. (2016) split the RecSys dataset in such a way that the last day of observations becomes the test data. We will apply the same approach to compare results with this previous research. This also includes removing items from the test data that are not found in the training set, as these items will have no influence on the trained model.

To validate the performance of the model during training, we propose using a three-split. The new validation set acts as observations that have not been trained upon. The validation set is used to tune the model using its hyperparameters and, in particular,

[6]Holdout data is simply data that has not been applied to the model during training or tuning. Tuning is usually done with a validation set.



Source: Own elaboration.
Figure 6.6: Data is split into training, validation and testing partitions.

avoid overfitting. Tuning the model using a validation set makes the validation set unfit as a test set, as the test set must be truly unknown. While the last day of observations acts as the test set, the day before the last day will act as the validation set. Figure 6.6 is a simple sketch of the two and three splitting method.

The sizes of the RecSys validation and test set is around 0.55% of the full dataset. Normally, the training, validation and testing set would be split around 60/20/20, respectively. However, having data this large makes splits of 98.9/0.55/0.55 sufficient when also considering the fact that we seek to serve recommendations covering trends at the most recent point in time, rather than historical trends.

When the model learns while training, batches[7] are usually served randomly. However, due to the temporal changes of shopping behavior, we propose feeding the model according to ascending age. Preferences of users can change due to sale offers, day of week, etc., which may ultimately affect the model performance. Sorting the input by time may help in adjusting weight and biases accordingly with time and accounting for trends and temporal changes that may occur. This process is not normal, however; with this, we argue that performance of recommender systems increases by tuning them towards newer data/time.

6.5 Augmentation Techniques

Click-stream sessions vary in length. Users may view items by mistake or search for information that affects their behavior. Recommender systems should be able to make accurate recommendations to users, regardless of their session lengths.

In the area of machine learning, image recognition techniques have been outperforming themselves recently by the use of augmenting the available data (DBL, 2015). These augmentation techniques were developed to overcome problems with the shortage of labeled image datasets. The basic idea of data augmentation is to make small changes to input

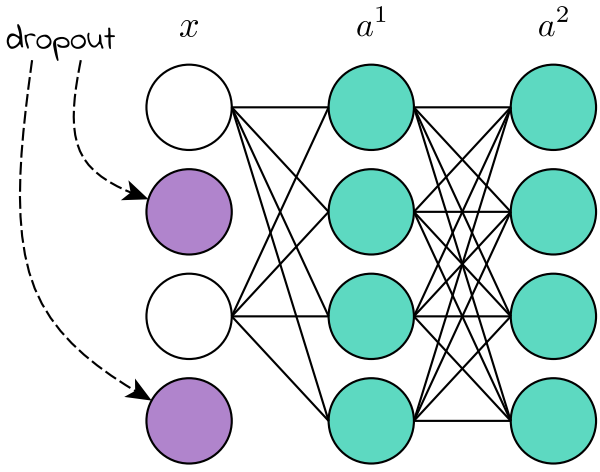
[7] A single forward/backward propagation training iteration.

data and reuse this as a new input. Within image recognition, these small augmentations would include editing contrast or exposure of input images. Eventually, these augmentation techniques allow for better predictions when feeding the models holdout data (Zhang et al., 2016).

This research is not based on images, but we argue that the same concept can be applied when working with sequential data. We seek to test four different ways of augmenting available data. This may ultimately help in improving performance of a deep learning recommender system.

6.5.1 Input Dropout

One way of augmenting the data is by removing (dropping) parts of the input while training the model. Within machine learning, this concept is often referred to as *dropout*. Dropout is a regularization technique and by applying it to the input layer, it also becomes an augmentation technique, as the actual data is augmented. Input dropout will affect session data by removing a chosen percentage of steps at random (setting values to zero). A basic example of dropping inputs is illustrated in figure 6.7. Purple nodes illustrate the dropped nodes while the turquoise remain intact. While dropout can be of use while training the model, it is never applied when testing the model on validation data or testing data.



Source: Own elaboration.
Figure 6.7: Applying dropout at the input layer.

In this research, input has been embedded, hence dropout is actually also applied to the embedded input of the model (also called, embedding dropout). Embedding dropout is seldom used, but has been proven to improve the performance by regularizing the data and avoiding overfitting (Gal and Ghahramani, 2016). Dropout in the input layer can be understood as dropping an event at random and teaching the model not to depend on specific events when updating weights and biases. This process will result in slightly

different input sessions at each epoch[8] because of randomly dropped events. This should make the model less sensitive towards steps that were not intentional or were not of interest as different combinations of a session get trained (Gal and Ghahramani, 2016).

6.5.2 Reversing Sequences

This technique simply reverses sessions and concatenates the reversed sessions with the original input sessions. A user's journey through the items could be relevant in both directions. A user interested in Harry Potter books at timestep t_1 may visit Harry Potter movies at t_2. We argue that the reverse behavior could be of interest for another user.

Compared to embedding dropout, this data augmentation is implemented in the pre-processing part and the reversed sessions are appended to the original training data.

6.5.3 Session Splitting

Proposed by de Brebisson et al. (2015), this method focuses on the number of timesteps any user would need to find the desired item. The idea is splitting the original session into new sessions of all possible timestep lengths. Given an input session:

$$[x_1, x_2, \cdots, x_n]$$

a number of n new sessions will be generated:

$$[x_1], [x_1, x_2], \cdots, [x_1, x_2, \cdots, x_n]$$

For each original session, the number of sessions generated is equal to the number of timesteps in the original session.

$$\begin{aligned} 1^{\text{st}} \text{ session} &= [x_1] \\ 2^{\text{nd}} \text{ session} &= [x_1, x_2] \\ &\vdots \\ n^{\text{th}} \text{ session} &= [x_1, x_2, \cdots, x_n] \end{aligned} \tag{6.1}$$

Corresponding y labels are generated as well (de Brebisson et al., 2015).

Splitting the original training set according to equation 6.1 will increase the size of the dataset, hence training time will likewise increase. The increased training time will be taken into account when considering and evaluating the best model.

[8] One forward/backward propagation of all training examples.

6.5.4 Briefly Viewed Items

We argue that events lasting less than five seconds are either accidental or not of interest to the user (Gal and Ghahramani, 2016). We can therefore classify these events as noise and remove them.

6.5.5 Temporal Training

As discussed earlier, we argue that time plays a role in the data, as trends come and go. Recent events in the data may therefore influence the preferences of users, hence recent sessions should weigh more while training the final model.

Classification often requires balanced datasets to be equally capable of classifying different classes. An unbalanced dataset may result in a model obtaining a better accuracy by predicting the majority class (Pozzolo et al.). Although this model and data differ from its original use, we argue that the dataset suffers from an unbalanced problem weighted towards old data.

This proposed augmentation works by running 1-2 epochs on the entire training set. Afterwards, data weighted towards newer sessions is applied. We will be applying the following weights on the training data according to time: $\frac{1}{1024}, \frac{1}{256}, \frac{1}{64}, \frac{1}{16}, \frac{1}{4}, \frac{1}{1}$. This will leave the most recent sessions intact, while only keeping a smaller portion of less recent sessions.

6.6 Summary

This chapter introduces the data behind the research: namely session data. Two datasets from e-commerce webstores are assessed: RecSys and AVM. Both dataset tend to be sparse, leaving many items less visited and few items very popular. The RecSys data includes a high number of events and sessions, whereas AVM includes fewer events and sessions. Furthermore, AVM has a much larger range of items than RecSys. Cleaning the data includes removing observations, such as sessions with only a single event and robotic behavior. Cleaning is necessary as the quality of the data plays an essential role when training upon the data.

We introduce padding as a way of preparing the data for recurrent neural networks which only take fixed sized inputs. Padding helps in expanding the session with zeros, allowing the recurrent neural network to train upon the data. Masking removes the effect of the padded zeros, leaving the accuracy unaffected.

Data augmentation techniques are introduced. These are ways of augmenting the data so that the model can train upon data that differs slightly from the original data. These augmentations may lead to better performance when training artificial neural networks.

CHAPTER 7

Models

> *"Machine learning algorithms frequently require careful tuning of model hyperparameters, regularization terms, and optimization parameters. Unfortunately, this tuning is often a "black art" that requires expert experience, unwritten rules of thumb, or sometimes brute-force search."*
>
> Snoek et al. (2012)

Building a recurrent neural network capable of handling this type of data is a complex task in numerous ways. It has been difficult to find inspiration from previous research, as deep learning for recommender systems remains a novel area. Figuring out how to approach and structure the building of such a model has been the main time-consuming task of the research project. We decided on building models using Google Tensorflow, a recently published framework[1] for building deep learning models in Python.

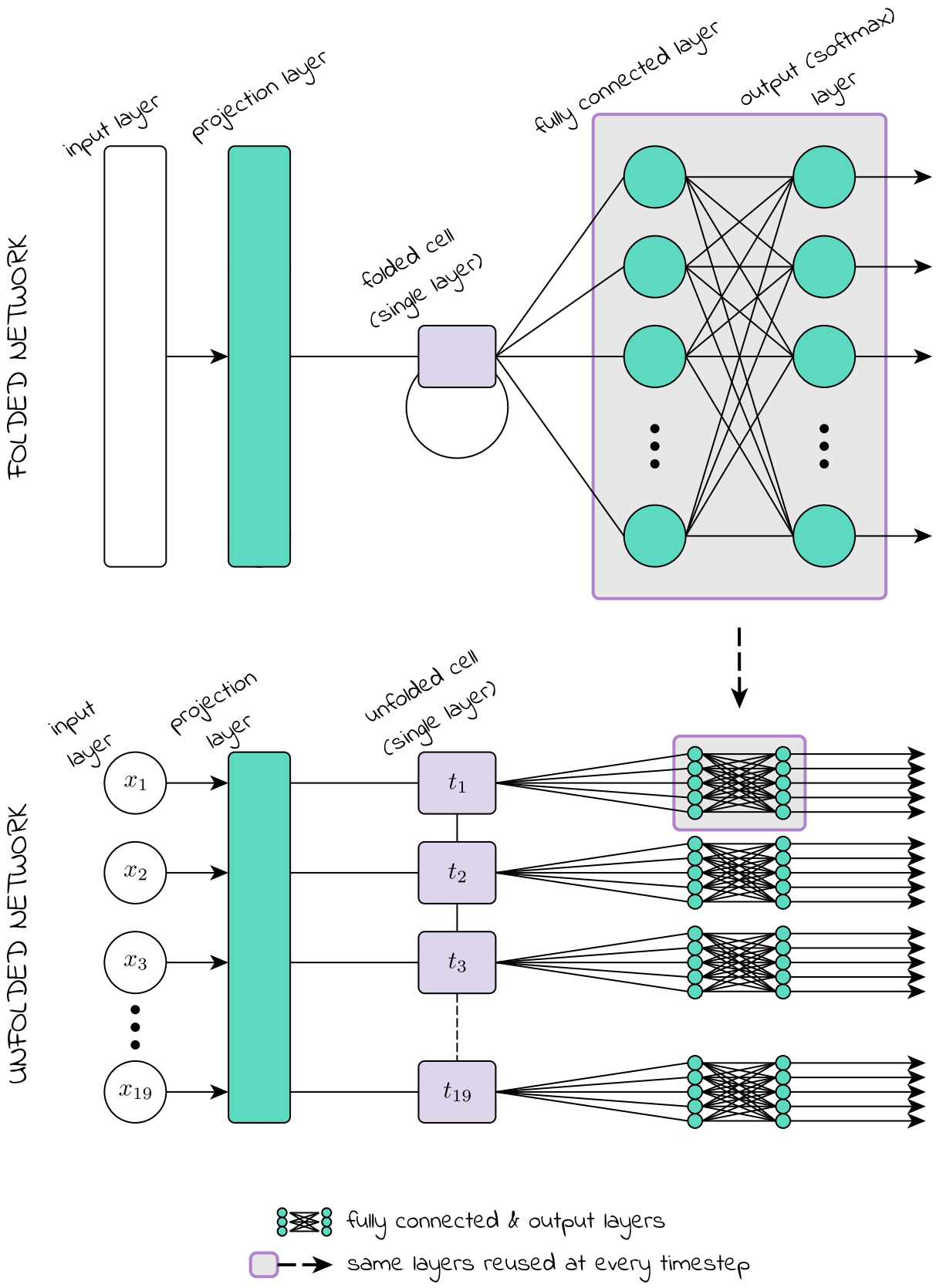
This chapter covers most aspects of the models that predict user recommendations on behalf of the RecSys and AVM datasets. We explain the proposed models, including the architectures, hyperparameters, loss function and the optimizer that yielded the best results.

7.1 Network Architecture

When developing the architecture of the model, we will be addressing critical challenges. Challenges include categorical input data and the handling of large product inventories. The architecture is based on different layers, each with a specific purpose that addresses these challenges.

In figure 7.1, the final architecture of the proposed models is shown; we will briefly cover the different aspects hereof.

[1] Tensorflow is a deep learning framework API for python. It allows for building and training deep learning models using GPU resources, which accelerates training times considerably.



Source: Own elaboration.

Figure 7.1: Architecture of the proposed models.

First, sequences serve as input to the model through the input layer. The embedded layer projects items of the input sequences into item vectors, which are used as inputs to the recurrent cells — the cells that handle sequential data over time, t. A fully connected layer transforms the cell outputs into logits that a softmax layer squashes into probabilities. These probabilities serve as the outputs of the model and are used to calculate the final accuracy of the model.

The rest of the chapter goes into detail about this architecture and the functionality of the model. Please refer to figure 7.1 at any time throughout this chapter.

7.1.0.1 Embedding Projection

As mentioned previously, neural networks are not suited for categorical input data, such as item IDs. Often one-hot encoding is applied to transform categorical data into a format that works with classification algorithms. Each row of a one-hot encoded matrix represents a category, i.e. an item. One representation of an item may look like: $[0, 0, \cdots, 0, 1, 0, \cdots, 0, 0]$, and have a total length of n, where n is the total number of items. However, one-hot encoding gets expensive as n increases. In this case, n is 37,483 for RecSys and 138,579 for AVM. Consequently, the one-hot encoding matrix would expand, requiring computational performance beyond current resources.

Instead, we will look to techniques utilized in NLP. Considering a sequence of items the same as a sentence of words, we understand that items equal words. Both elements have similar characteristics and both represent a specific meaning in sequences. In NLP, embedding techniques are used to vectorize words based on features of each word. A layer that embeds the categorical data, such as words or items, into machine computational values plays an important role (Bengio et al., 2013). In our case, each item (word) becomes a vector representation of features that characterize the item. If one was to consider features of an item, one would consider color, type, shape as well as semantic features. With embedding, we rely on the learning algorithm to uncover such features of each item[2] (Blitzer et al., 2004). Like the neural network, the idea of representing a word or item as a vector of feature values comes from our brain. We learn and use distributed representations to understand new objects by linking them with known objects with similar characteristics (Mikolov et al., 2013).

The algorithm basically learns to link each item to a vector representation of continuous values. Each item exists as a point in a feature space; each dimension in this space will represent a characteristic of the item and similar items should be closer to each other in this feature space. The number of dimensions of the vectorized values is manually set and remains the same for all items (Mikolov et al., 2013).

With n amount of items, the embedded feature space will be smaller than that of

[2]Once a model is fully trained, we are left with fully trained item embeddings.

a one-hot encoding matrix and therefore more computationally efficient. While a one-hot encoded item vector length equals n, an embedded item vector size equals the chosen amount of embedding dimensions. With RecSys and AVM, this reduces the dimensionality by a factor of over 100.

In section 8.5 we will visualize the fully trained item embeddings using t-SNE[3]. This will give an indication of expected item neighborhoods in a two-dimensional space.

7.1.1 Recurrent Cells

The models were built using the recurrent cells as covered in section 4.3.2.1 and 4.3.2.2. By training upon both the LSTM and GRU architectures, we ensure to add the best cell to the model by comparing the accuracy of networks with LSTM and GRU.

Sequences of items works as input to cells. The network will have n number of cells, where n equals the total timesteps and each cell accounts for one item[4]. As briefly covered in sections 4.3.2.1 and 4.3.2.2, the inner workings of the cells account for the memory of the network, as the cells decide what to store in memory. This means that the first item can affect the next item, which is what makes these cells suited for sequential data.

By stacking cells on top of each other, the network captures the hierarchical structure of sequences. Setting the number of recurrent cell layers means adjusting a hyperparameter of a recurrent neural network. Researchers previously tested two-layer recurrent cells on the RecSys dataset and concluded that no improvement was made (Hidasi et al., 2016). Testing of multiple layers of recurrent cells is therefore not tested during this project. This will free time to tune other hyperparameters.

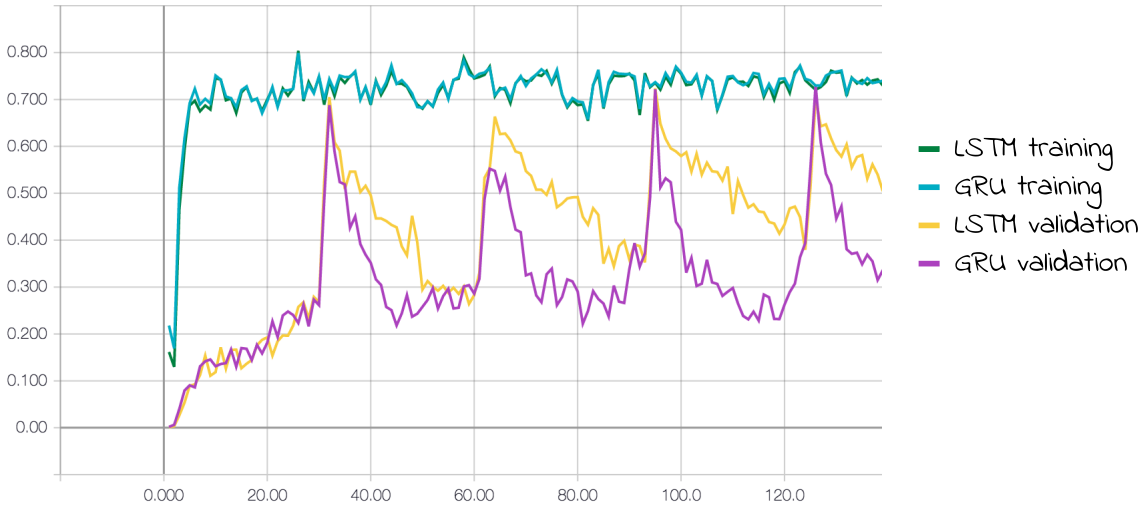
7.1.1.1 Best Cell Architecture for User Behavior

While both the architectures LSTM and GRU are highly useful when training models upon sequential data with a historical trend, they can differ in results. We examine the different outcomes of applying both architectures on the RecSys dataset using a basic model built without dropout and without data augmentation. Figure 7.2[5] compares the accuracy scores of LSTM and GRU. As previously discussed in section 7.1.1, we expect that both GRU and LSTM have equal scores. We can confirm this expectation, when considering the training scores of figure 7.2. However, LSTM tends to capture more historical trends, when considering the validation scores. We see that, for a start, both architectures learn equally well, considering the validation data. After the first peak (first epoch), GRU seems to quickly forget old trends, lowering the accuracy of GRU. As discussed in section 4.3.2.1, we see that the long-term memory states of the LSTM architecture tends to keep historical

[3]t-SNE will be covered briefly in section 8.5.

[4]When items act as input to cells, the items are vectorized.

[5]X-axis values, x, times 500 equals the batch number.



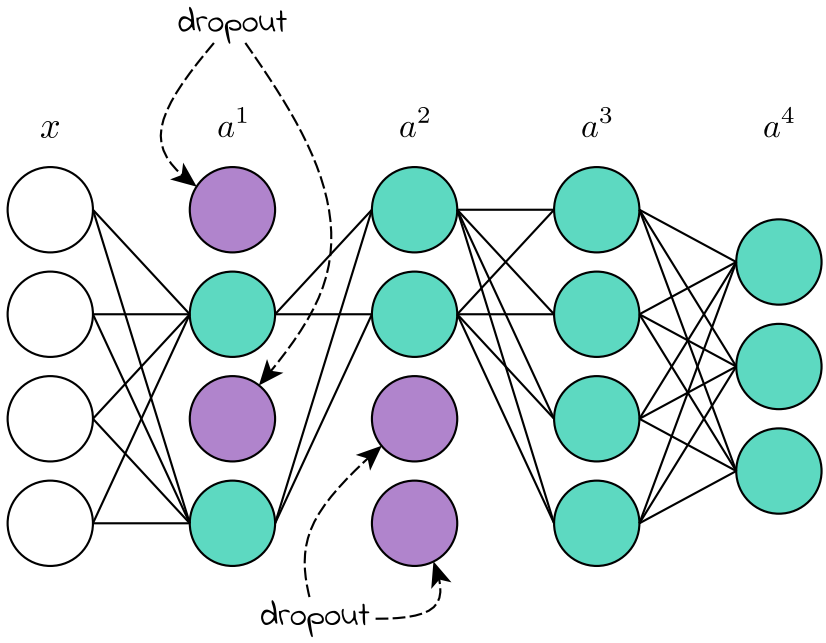
Source: Own elaboration from Tensorboard.
Figure 7.2: Evaluating LSTM and GRU scores on RecSys data.

trends in memory, resulting in higher accuracy. The findings of this test motivates the use of LSTM when training and tuning models. Recurrent neural networks discussed throughout the rest of the thesis will be built using the LSTM architecture.

7.1.1.2 Dropout in Recurrent Units

Recurrent neural networks are models of high flexibility and have the capabilities to fit a wide range of nonlinear patterns. However, this type of network also tends to overfit data, leading to underperformance on the test set and in practice (Kingma et al., 2015). Fortunately, regularization techniques can mitigate the overfitting problem. As previously presented in section 6.5.1, dropout is a popular regularization technique. While in input dropout, events are dropped, dropout in recurrent units differs slightly. If dropout is applied, each node is subject to being switched off during training. Dropout is always applied with a chosen probability — the keep probability, p (also referred to as the dropout rate, $1 - p$). This is another adjustable hyperparameter. Applying dropout to a single layer of a neural network with $p = 0.5$ means nodes will be switched off 50% of the time. By dropping a node, it means temporarily removing the node with its input and output connections (Molchanov et al., 2017; Srivastava et al., 2014). Figure 7.3 is a simple example of dropout inside the network. Note that connections to dropped nodes are removed, resulting in a new architecture of the network at each iteration.

New research suggests how dropout can improve results when training large recurrent neural networks by implementing variational dropout (Gal and Ghahramani, 2016). This method drops the same network nodes at each timestep, while randomly dropping cell inputs, recurrent connections and cell outputs. Without variational dropout, dropout simply drops nodes at random but leaves recurrent connections undropped. Vari-



Source: Own elaboration.
Figure 7.3: RNN dropout.

ational dropout even allows for individual keep probabilities for inputs, states and outputs (Molchanov et al., 2017). In theory, this leaves us with more tuning abilities to improve results; however, due to lack of time, a single keep probability is used for all.

7.1.2 Fully Connected Layer

The updated recurrent neural networks cells produce the input of the fully connected layer at every timestep. The unfolded network in figure 7.1 illustrates the same fully connected and output layers being a part of the network at every timestep. The fully connected layer is a single layer connecting all inputs to all nodes of the layer. This layer will have the same number of nodes as there are unique items. Hence, each node in the fully connected layer represents an item.

The activations of the fully connected layer form logits[6] that are passed on as input to the softmax layer. A logit value is computed for each node. Nodes outputting higher logit values represent items that may be good candidates for the final predictions. The importance of the logit value does not lie in its specific value, but in its relative size.

7.1.3 Softmax Output Layer

The final layer of the architecture in figure 7.1 is a softmax layer. We discussed softmax previously in section 4.2.4.1 and found that softmax takes an input vector and squashes

[6]Logarithmic values.

these inputs into a probability distribution of multiple classes.

First, this final layer, L, receives inputs $\boldsymbol{z}^L$ from the previous layer. We then apply the softmax function to any input value, z_j^L (here, the j^{th} node in layer L). This will give us the activation value of the node, as previously discussed and listed in table 4.1, where $\sigma(x) = a_j^L$. The softmax function normalizes the activation logits from the fully connected layer into probabilities. Since the total of the softmax outputs is equal to one, we may interpret these outputs as probabilities. In section 5.2, we presented how the accuracy performance of a network is calculated. We consider the top 20 items according to the probability rate. Computing the accuracy happens by comparing the top 20 predictions with the actual target values and deriving the number of correct predictions. A true prediction is when one of the top 20 items is equal to the target item and a false prediction is when none of the 20 items were equal to the target item.

7.2 Learning Behavioral Patterns

We have introduced the different aspects of the architecture of the models. This section focuses on training the network. We seek to cover all techniques applied when training the network, making it learn behavioral patterns in the RecSys and AVM datasets. Throughout chapters 3 and 4 we cover most aspects of artificial neural networks and recurrent cells. In this section, the motivations behind the chosen methods and techniques will be addressed. The depth of technical detail varies and depends on the specific topic.

First, we briefly discuss the applied loss function. Second, we cover the basics of the inner workings of the applied optimization algorithm, whose job it is to minimize the value of the loss function. Last, we introduce all considerable hyperparameters of the network in question.

7.2.1 Loss Optimization

By now, we have covered the main architecture of the network from the inputs $\boldsymbol{x}$ to the final output $\boldsymbol{a}^L$. We know from section 4.2.3, that our network can be updated using the back propagation algorithm. For the network to know if predictions are right or wrong, we apply a loss function.

7.2.1.1 Loss Function

This network utilized the cross-entropy loss function based on the softmax activations. Cross-entropy is applicable due to its purpose of measuring similarity between the true targets, y, and given computed predictions (activations of the softmax layer, $\boldsymbol{a}^L$)). The

loss, denoted as cost C, is given by:

$$C = -\frac{1}{n}\sum_{x}\sum_{j}[y_j \ln a_j^L + (1 - y_j)\ln(1 - a_j^L)] \tag{7.1}$$

Cross-entropy is suitable as a loss function mainly due to two reasons: C is always positive and C should gradually decrease and ultimately hit zero as the distance between predictions and targets becomes smaller. Likewise, the loss C is high if the distances between predictions and targets are wide. Once C has been calculated, we use this loss combined with an optimizer and the back propagation equations, 4.7 and 4.8 in particular, to update biases and weights when propagating backward through the network. Ultimately, the next training iteration will result in a smaller loss as weights are adjusted.

7.2.1.2 Optimization Algorithm

Based on recent research by Kingma and Ba (2014), we want to utilize ADAM, a method for efficient stochastic optimization of the loss. ADAM will not be covered in detail, nor will we discuss the algorithm itself. This section rather focuses on the motivations behind the choice of ADAM as the optimization algorithm. For the ADAM algorithm, please refer to appendix A.

As covered in section 4.2.1.2, stochastic gradient descent reduces the cost of running back propagation in neural networks. ADAM is a novel way of utilizing stochastic gradient descent when optimizing a neural network. In particular, ADAM was worth considering as this algorithm focuses on optimizing high-dimensional parameter spaces.

ADAM can find the minimum[7] in very few steps, which is an advantage when working with large datasets such as RecSys and AVM. This saves us time, and allows for the testing and tuning of other hyperparameters. Another important aspect of the ADAM algorithm is its adaptive adjustment of the step sizes[8]; ADAM will start out with relatively high step size and decrease the step size as a minimum is reached (Kingma and Ba, 2014).

The architecture of the network has been presented in the previous sections. The following sections will cover tuning of hyperparameters — an essential part of training artificial neural networks.

7.2.2 Tuning Hyperparameters

Any neural network architecture requires tuning of hyperparameters to maximize performance. These hyperparameters can be chosen by reason, but may require tuning through empirical testing to improve the performance of the model, as also stated by Snoek et al. (2012) in the introductory quotation. Training times for the models in question take

[7]The optimizer searches for minimums, i.e. where the loss is lowest.
[8]Steps for which the learning rate decides the size.

between 8 and 17 hours each for both the RecSys and the AVM models, depending on the setting of hyperparameters. Once the model was built, there was limited time to tune the hyperparameters. Considering the training time, a grid-search of the best hyperparameters for each model was therefore not a possibility. The applied approach included stepwise testing of hyperparameters and research on good choices of initial values of hyperparameters.

Learning Rate As previously discussed, the learning rate is used by the ADAM optimizer to define the step size towards a minimum. Regarding the RecSys dataset, tests were run with learning rates ranging from 0.01 to 0.001. We found that a learning rate of 0.0025 is optimal. For the AVM dataset, the learning rate is lowered to 0.0020 due to a smaller batch size, consequently resulting in more updates of weights and biases. The learning rate will be assessed below when discussing the batch size.

Dimensions of Embeddings When tuning the embedding projection layer, we aim to find the optimal number of dimensions[9] that represent the items in the best possible way — refer to section 7.1.0.1 for details. Tests on the RecSys dataset was made with the number of dimensions ranging from 50 up to 1,000. As the number of dimensions increases, the training time increases as well. Therefore this, like other hyperparameters, is also a tradeoff between a good choice of hyperparameter value and training models in a shorter time. An embedding layer with 300 dimensions yielded the best results regarding the accuracy of the models.

Accuracy of the RecSys model increased as the number of dimensions increased up until around 300 dimensions. The reason why the accuracy stopped increasing was most likely due to item features extracted from the user behavior not having more than 300 differences. Higher dimensionality would result in less accurate representations of the items and thereby a worse accuracy of the full model as the input quality decreases. Hence we assume that 300 dimensions represents the item feature of an e-commerce site with over 30,000 different products.

Size of Recurrent Cells The number of nodes usually affects the model performance greatly, as too low a number of nodes will make models underfit, while too many nodes may cause overfitting (Xu and Chen, 2008). A rule of thumb states that a number of nodes between the number of inputs and outputs is optimal (Berry and Linoff, 1997). We found that setting this hyperparameter to 500 was too time consuming for training. We ended up testing values in the range of 100 to 500 and found that the optimal number of nodes in the recurrent cells was 200. This hyperparameter tends to affect computation heavily, as each added node brings along

[9]Dimensions of embeddings are also called features.

weights and biases — subjects of updates at each forward/backward propagation iteration.

Keep Probability of Dropout in Recurrent Cells The dropout method applied to the recurrent neural network allows individual dropout rates, but because of the limited time, individual dropout rates are not tested. Dropout rates range from 0.25 to 1.00 to find the best setting that regularizes for overfitting.

Surprisingly, for RecSys no dropout, i.e. keep probability is set to 1.00, gave us the winning result. This was a surprise, but may be due to important updates to weights and biases because of temporal changes. Applying dropout slows this process and thereby hinders the models in learning the temporal changes. For AVM, a dropout keep probability of 0.35 yielded the best results. Since the AVM dataset was reduced in size, it seems that the AVM model easily overfits the data. Dropout in the recurrent cells regularizes for the overfitting effect, increasing the validation accuracy of the model. Please refer to chapter 8 for the final results of models.

Batch Size As previously introduced in section 4.2.3, we know that training a neural network includes updating weights and biases by propagating backward through the network. The size of the batch determines the number of sessions included in each of the forward/backward propagation iterations. As discussed in section 7.2.1.2, each iteration takes a step towards the lowest loss point, where the distance between predictions and actual targets is at the lowest point. The learning rate determines the size of this step. This means that the learning rate and the batch size affects each other. A small batch size will increase the number of iterations (updates) of the network, allowing for a smaller learning rate with more steps, while a big batch size decreases the number of updates (Li et al., 2014).

Big batch sizes (between 256 and 1024) allow for faster training due to exploitation of GPUs[10] (Li et al., 2014). When training the models in question, GPUs were available, allowing for a batch sizes of 512 sessions for RecSys and 256 sessions for AVM. Accordingly, learning rates were adjusted for both RecSys (0.0025) and AVM (0.0020).

Epochs An epoch is a single iteration through the training data and the number of epochs denotes the number of times the model should iterate through the training data. By iterating through the same data, more steps are taken towards a minimum and weights can be fine-tuned (Raschka, 2015).

[10]Graphics processing unit, designed for computing thousands of small tasks at once.

The models developed for AVM include dropout. Any layer with dropout will differ at each epoch iteration, which means that the data changes in format at every epoch iteration.

Basically, most models were trained until the validation statistics stopped improving. For the RecSys model, we found that after just 1-3 epochs, depending on data augmentation techniques, the validation statistics reached their maximum. However, testing on different dropout keep probabilities and learning rates seems to require more than five epochs, as learning was slowed. The AVM model required up to 50 epochs.

For reference, table 7.1 lists the discussed values of hyperparameters for both models.

Table 7.1: Summary of Hyperparameters

Hyperparameter	**RecSys**	**AVM**
Learning rate	0.0025	0.0020
Number of embedding dimensions	300	300
Number of nodes in cells	200	200
Recurrent dropout keep probability	1.00	0.35
Batch size	512	256
Number of epochs	3	50

7.2.3 Masking

As previously described, recurrent neural networks demands input of fixed lengths. Since sessions vary in length, we found that techniques of natural language processing may be of assistance. When sentences vary in length, masking, a NLP technique, can be applied. When preprocessing the data, session sequences were padded with zeros in a such way that all sessions were equal in length. Padding solves the problem of the input length, but yielded new issues. Zero paddings need to be assessed when training the model to avoid overfitting and false predictions; the model sees zeros as an item, consequently making the model predict zero at any time due to the amount of padding needed.

Therefore, the model is explicitly told to ignore padding with the use of masking. Equation 7.2 exemplifies masking by showing the Hadamard products of two matrices; the mask matrix of zeros and ones and the matrix of values and paddings.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 1.2 & 0.3 & 2.4 & 2.3 & 1.4 \\ 1.9 & 1.1 & 0.2 & 3.1 & 0.5 \\ 0.6 & 0.3 & 1.3 & 0.7 & 0.9 \\ 0.5 & 1.3 & 2.4 & 3.5 & 1.2 \end{bmatrix} = \begin{bmatrix} 1.2 & 0.3 & 2.4 & 0 & 0 \\ 1.9 & 1.1 & 0.2 & 3.1 & 0.5 \\ 0.6 & 0.3 & 0 & 0 & 0 \\ 0.5 & 1.3 & 2.4 & 3.5 & 0 \end{bmatrix} \tag{7.2}$$

The concept of equation 7.2 is the same for our models. First, a masking matrix is generated based upon the input matrix with sizes $m \times n$, where m is the total number of sessions and n is the number of timesteps. The masking matrix holds zeros and ones. If, for example, the second cell of the input matrix, a_{12}, is a real value, the cell, b_{12}, of the masking matrix would be one; otherwise it would be zero.

Looking at figure 7.1, the softmax layer outputs a two-dimensional matrix of sizes $m \times n$. Using the Hadamard product of the two matrices, the loss cells multiplied with either a zero or one, depending on the value being a padding or not, will remove the loss of padded values. Removing the losses of the padded value will affect how the model updates weight and biases through back propagation, since it no longer adjusts errors of padded values (Neubig, 2017).

This method works the same way for the accuracy rate; however, accuracy works in a three-dimensional space and the matrix keeping padding information must therefore be in a three-dimensional space as well. Using the Hadamard product of the both three-dimensional space matrices will ensure that all padded values' prediction is set to false. Considering equation 5.1, padded values count as false predictions in vector $\boldsymbol{p}$. However, they still increase total n predictions, hence the number of paddings is subtracted from the denominator of equation 5.1.

To make the computation more efficient, allocating additional memory space for paddings happens at each training batch. By allocating memory at each batch, the memory demand drastically decreases as the matrix size becomes the product of *batch size*, *timesteps* and *top 20 predictions* rather than the product of *sessions*, *timesteps* and *top 20 predictions*.

7.3 Summary

This chapter presents the network architecture developed throughout the research project. The network is built with input, embedding, recurrent, fully connected and softmax layers. First, we assess the issue of feeding categorical data to the network. This issue can be solved by applying embeddings; generating vector representations of items. These vectors are used as input to recurrent cells capturing historical information from sequences of items. We tested two structures, LSTM and GRU, and found that LSTM performs best on the RecSys dataset as LSTM tends to have better memory further back in time. Cells output the input of the fully connected layer. Each node of the fully connected layer represents an item and activations of this layer represent the logits that are fed to the softmax. Softmax produces a final probability distribution over all items.

The optimization algorithm ADAM uses the loss of the cross-entropy and stochastic gradient descent to update weights and biases, while back propagating through the

network.

Finally, we cover the inner workings of masking and how this helps in separating padded values of shorter sessions from actual values. Masking makes it possible to calculate the correct accuracy and loss, without interference from padded values.

Part IV

Findings

CHAPTER 8

Model Performance Evaluation

In this chapter we present the models developed throughout this research. New and previously acknowledged data augmentation techniques will be utilized to determine, if these improve performance of models using the RecSys and AVM datasets. When applying the data augmentation techniques, the hyperparameters will remain the same as stated in chapter 7. Due to rather long periods of training, tuning from scratch with multiple data augmentation techniques is impossible within the given time frame.

Besides comparing models with each other, we benchmark all models with the baseline model. The comparison will, in particular, focus on, but is not limited to, the accuracy of the models. Novelty, serendipity, and diversity, all play an important role in providing the users with the correct information. Table 8.1 shows a list of all models and their respective data augmentation techniques. Any models based on data augmentation are written in bold. The following sections present results of the fully trained models in terms of accuracy, novelty, serendipity, and diversity with regards to both datasets. Finally, we briefly illustrate a t-SNE visualization of the trained item embeddings.

Table 8.1: List of Modeling Techniques

Short Name	Specification
Basic RNN	Basic RNN without data augmentation.
RNN ID	RNN with input dropout.
RNN RS	RNN with reserve sequences added to original input sequences.
RNN SS	RNN with split sessions input.
RNN BV	RNN with input excluding items viewed only for a brief period of time.
RNN TT	RNN with temporal training focused data added to input data.

8.1 Accuracy

Accuracy is the first and most important evaluation metric. Consequences of low accuracy may include irrelevant recommendations. Novelty, diversity, or serendipity of recommen-

Table 8.2: List of Accuracies

Model	RecSys	AVM	Comment
Baseline	50.65 %	20.18 %	
Hidasi et al. (2016)	63.22 %	—	RNN with 1,000 neurons
Tan et al. (2016)	71.29 %	—	RNN with 100 neurons
Basic RNN	**71.55 %**	50.43 %	No data augmentation
RNN ID	71.19 %	50.68 %	
RNN RS	69.02 %	**52.85 %**	
RNN SS	24.16 %	—	Low RecSys acc. — AVM omitted.
RNN BV	71.32 %	50.00 %	
RNN TT	71.48 %	50.10 %	
RNN ID + RS	—	52.79 %	RecSys omitted.

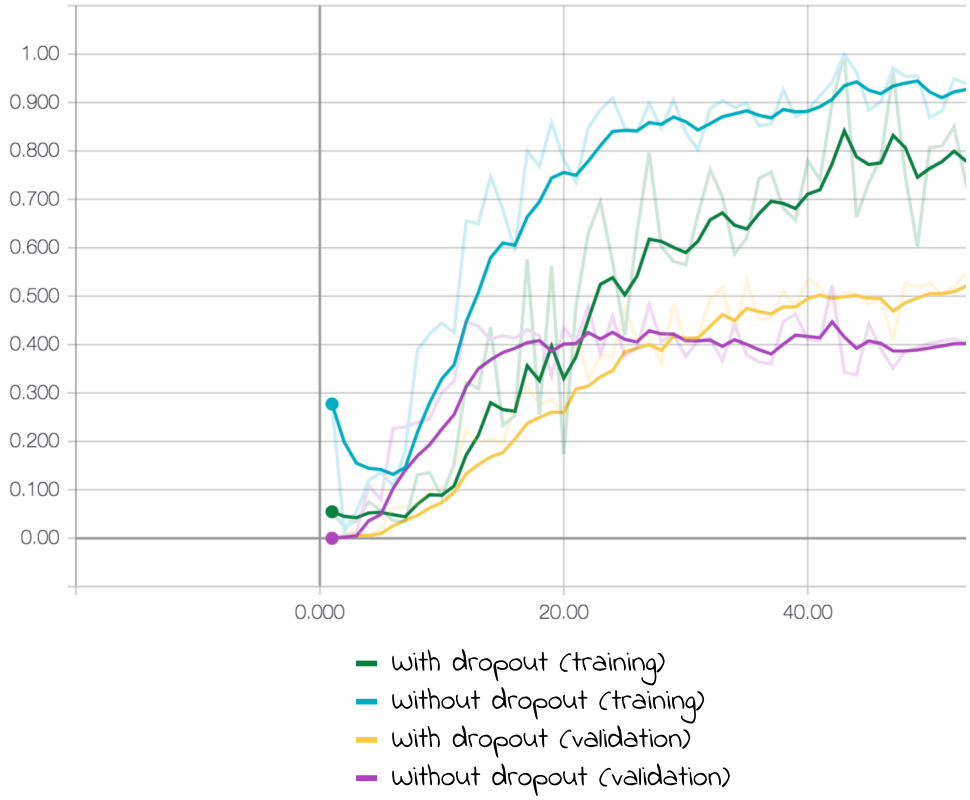
dations does not matter, if the recommendation is already irrelevant for the user. After diving into the accuracy performances, we will be addressing the abilities of the models to overcome challenges presented in section 3.3. Accuracies of all models are listed in table 8.2.

The baseline model score an accuracy of 50.65% on RecSys and 20.18% on AVM. As discussed in section 6.3, the AVM dataset is rather sparse. We see that the accuracy of the kNN baseline model decreases as the amount of items increases, while the sessions remain relatively few. Clearly, the combination of more items and fewer events decreases the performance of both the baseline and the proposed models. However, the performance increase on AVM is much higher compared to that of RecSys. This supports theory of recurrent neural networks being favorable models when modeling sparse data. Looking at the baseline model, we see that the accuracies of RecSys and AVM were improved by 41.3% and 161.9%, respectively. We find these improvements in accuracy satisfactory.

The results of the recurrent neural networks are almost identical for the RecSys dataset, but accuracy seems to slightly decrease with the data augmentation techniques, whereas, the AVM models improve performances with the augmentation techniques RNN ID and RNN RS. The proposed data augmentation techniques' effects on the models are addressed in the following.

Basic RNN As described in section 7.2.2, performance was not increased by applying dropout in recurrent cells on the RecSys data. Most recurrent neural networks tend to overfit on noise in the training set without finding the latent factors (Gal and Ghahramani, 2016). In figure 7.2 on page 63 we saw that the Basic RNN avoids overfitting the data, which can be due to trends and temporal changes. The model must continuously update weight and biases accordingly with time. Hence, the model does not get stuck with the same popular items and noisy sessions. By applying dropout, the RecSys models learning ability are weakened and the accuracy decreases as the models cannot follow the fast changing trends.

The AVM data differs from the RecSys as AVM easily overfits the data. The AVM models require a low dropout keep probability of 0.35 to avoid overfitting. Consequently, each batch only keeps 35% of randomly chosen nodes, forcing the model to use all connections and not depend on specific connections between nodes. Figure 8.1[1] illustrates the Basic RNN models of AVM, with and without dropout in recurrent cells[2]. We see that trends occur, but to a limited extent. User behavior seems to change slowly. Applying dropout makes the model predictions rather precise on the training data, but the validation tends to lag behind. Finally, the test shows that accuracy rose from 40.06% without dropout to 50.43% with dropout, because the model fits to latent factors instead of noisy sequences of the training set.



Source: Own elaboration.

Figure 8.1: AVM Basic RNN compared with/without dropout in recurrent cells.

RNN ID Like the dropout in the recurrent units, input dropout also decreases accuracy on RecSys, whereas input dropout improves accuracy of AVM. Removing steps in the RecSys dataset seems too expensive and weakens the learning of the model. A dropout keep probability of 80% and 65% was applied, where 80% yielded best results on both RNN ID's.

[1] X-axis values, x, times 500 equals the batch number.

[2] For clarity, the figure illustrates smoothed averages. Muted lines represent the original scores.

RNN RS Doubling the data by reverting sequences seems to work well with AVM, but not for RecSys. We expect that the lower rate of temporal changes and a less amount of events was useful in training upon reverted sequences for AVM. As RNN RS doubles the amount of sessions, AVM enjoys an increase in accuracy of 4.80%[3] on the test set, due to a low events per item ratio of 18.56. RecSys had a better distribution of item events, which may cause a decrease in accuracy of 3.67%.

RNN SS Splitting the sessions into all possible lengths as proposed by de Brebisson et al. (2015), was only tested on the RecSys. Performance was so poor that training such model AVM is omitted. This technique emphasizes the first events in a session. The poor results are likely due to diversity and randomness in early events, as users may become specific later in sessions.

RNN BV This technique includes training models on a full dataset combined with a cleaned version omitting events with an event lasting for less than five seconds. This augmentation did not improve the Basic RNN accuracy. We expect that models learn these aspects on their own — whether an event is important or not. Consequently, RNN BV contributes no value to the model.

RNN TT This includes weighing the model towards recent sessions once first epochs has run through the full dataset. This did not improve the results of the Basic RNN. Since we already align data according to time, it seems that training of Basic RNN captures the temporal changes. We found that accuracy did not increase with weighting the training set towards recent sessions.

8.1.1 Mitigating Cold-start Problems

User-based memory models, as covered in section 3.1.2.1, require large amounts of data to make quality recommendations. Item-based memory models, such as the baseline models, are often useful as they compute recommendations before users visit. However, we argue that a user-based approach can overcome the cold-start problem with the developed recurrent neural networks. Even though most sessions are shorter than four events, accuracy of recurrent neural networks hits 71.55% for RecSys and 52.85% for AVM. This is a strong indication that the models build on user-based approaches suffers less from cold-start problems, when comparing with the item-based approaches. Table 8.2 supports this belief as models increases baseline accuracies by $41,3\%$ for RecSys and $161,9\%$ for AVM.

Using customer session data allows for quick predictions. Recommendations may tend to be less diverse than recommender systems build upon long customer relationships.

[3]Compared with Basic RNN.

However, they provide accurate and helpful recommendations that ultimately may convince new customers to become regular customers.

8.1.2 Temporal Changes

Temporal changes in users' preferences were a potential challenge when modeling RecSys and AVM. Usually, machine learning algorithms assume data to be independent and identically independent. When building recommender systems, these assumptions do not hold since new items continue to appear and the users change behavior according to trends (Tan et al., 2016). As we seek to generate recommendations for future users, accounting for these temporal changes plays an essential role.

Periods of months, weeks, or days can affect how well the models perform on new sessions. Popular recommender systems, like the baseline model, do not take these time sensitive factors into account. In fact, new users are likely to get outdated recommendations without relevance for the new users. However, it heavily depends on the specific case, e.g. clothes fashions changes quicker than children's toys.

The first recurrent neural network build upon session data, as proposed by Hidasi et al. (2016), did not account for temporal changes, when training the model. The accuracy hit 63.22%., as listed in table 8.2. Figure 7.2 on page 63 shows a clear trend of how the validation set performs much better when the models trains on recent data. Immediately, as new epochs begin, the accuracy drops, due to old data at the start of each epoch. Training the models on older data actually makes the performance worse, because it ends up focusing on out-dated factors. The best of the proposed models for RecSys have an accuracy of 71.55%, an improvement of 13.18% compared to the Hidasi et al. (2016) model.

8.1.3 Reactivity

In theory, recurrent neural networks should react quickly to changes in users' preferences. If the user behavior changes considerably, models will update hidden states of cells with more recent information and consequently forgetting older information. By emphasizing more new information, the recommendations served to users should match users' new direction. Non-sequential models do not account for changes in time; making these models less reactive. The traditional kNN looks at an item and recommend neighbor items. kNN react to changes in the users' direction as the algorithm only bases recommendations on current item. Therefore kNN also discards sequential information that may be important for new user directions.

8.1.4 Popularity Bias

Accounting for popularity bias was not a priority when building the models, but it was indirectly affected by the increase in accuracy. An accuracy rate of 71.55% and 52.85% shows the models' capability of making a wide range of item recommendations — not only items of high popularity.

Hidasi et al. (2016) tested how well recommendations (predictions) of *top 20 most visited items* performed on the RecSys dataset. This *popularity recommender system* only scored an accuracy of 0.5%, which shows that user specific preferences and the ability to recommend diverse items is crucial.

8.2 Novelty

As covered, accuracy is a key factor in providing users the support they need for making decisions using recommender systems. Even though the proposed models make far more accurate predictions than the traditional baseline recommender system, these new models must also account for other aspects, to ensure user satisfaction. Models makes recommendations based on previous user sessions; novel or diversity of recommendations depends on previous users' behavior. The item with the highest probability of being accurate may be too obvious and therefore not offer the customer any support. Item to item collaborative filtering techniques, such as the baseline, will often trap users in a *similarity hole* (Herlocker et al., 2004). Issues, such as too obvious, too similar or already seen items, can be of significance when supporting users. (McLaughlin and Herlocker, 2004).

We can help to make recommendations novel and diverse. Ensuring novelty in recommendations means providing users with new items, that users not previously either saw or knew about. Models should in theory not recommend items previously visited, e.g. if user a visited item i, item i should not be recommended user a. unless specifically telling the model to avoid a range of items (the previously visited items), this can become a minor issue. As stated previously, kNN only looks at the current items and would therefore also be subject to recommending previously visited items. In practice, a quick fix would be to simply check and omit previously visited items from the range of predictions served by the model, whether that is a recurrent neural network or kNN. As the models serve the 20 most likely items[4] and recommender systems in practice only show less than half of 20 items, a buffer of 10 or more items allows for filtering out previously visited items. This method is easy since we know the items a user already have visited. However, it gets more difficult when filtering items that the user is familiar with, but never visited and thereby leaves the recommender system without any feedback hereof. Instead, we argue that items

[4]Items with highest possibilities of being the next item the user visits.

of less popularity are more novel towards users than items of high popularity (L et al., 2012); item categories often tend to sort items according to popularity. Users who did not visit items of high popularity actively chose not to, due to lack of interest or previous knowledge about the items. Filtering out item of high popularity may help in reducing recommendations of user-familiar items.

8.3 Diversity

Making diverse recommendations increases the probability of providing support to users. Research has shown that user satisfaction correlates with providing diverse recommendations, thus exposing the users to a wider range of items (Ziegler et al., 2005). Item to item collaborative filtering methods, such as the baseline model, tend to find most similar items when generating recommendations. Hence, they reduce diversity to make more accurate predictions.

In theory, recurrent neural networks should outperform and recommend more diverse items, since these networks looks at behavior similarities between users, whereas the baseline looks at similarities between items. Ziegler et al. (2005) also showed that user based models makes more diverse recommendations than item based. Diversity in the behavior of a user increases the diversity of the recommendations. Meaning users viewing only one category of items, will get recommendations of limited diversity and vice versa.

8.3.1 Long Tail Items

As stated in section 6.3, items with less than five views — the long tail — can generate noise and consequently were dropped. It was a trade-off between having a higher accuracy and being able to recommend rarely seen items. As the purpose of these models was to make accurate recommendations for new users (with sparse feedback data), we argue that accuracy is of higher value than rarely viewed items. Including rarely viewed items would not ensure full diversity, as the weights and biases towards these may be small and thereby not contributing when predictions are made.

8.4 Serendipity

The goal of introducing serendipity into the system is to provide users with something else than what they were looking for. Recurrent neural networks should be able to give users the same experience, as if a person was visiting any shopping mall and happens to find some unexpected, but interesting, item.

The architecture of recurrent neural networks should provide more diverse results than

kNN. kNN is more prone to showing popular items, as it computes the probabilities based on items that other users have rated equally. Items with highest probabilities would be popular items — items with many events. Rather, recurrent neural networks looks at the sequence of events, accounting for time and sequential patterns. Therefore, the model should be capable of recommending items less related in terms of neighborhood.

Serendipity is linked with novelty and diversity, as they ensure new and different items. By accounting for novelty and diversity in post-processing, the probability of recommendations of serendipity increases likewise.

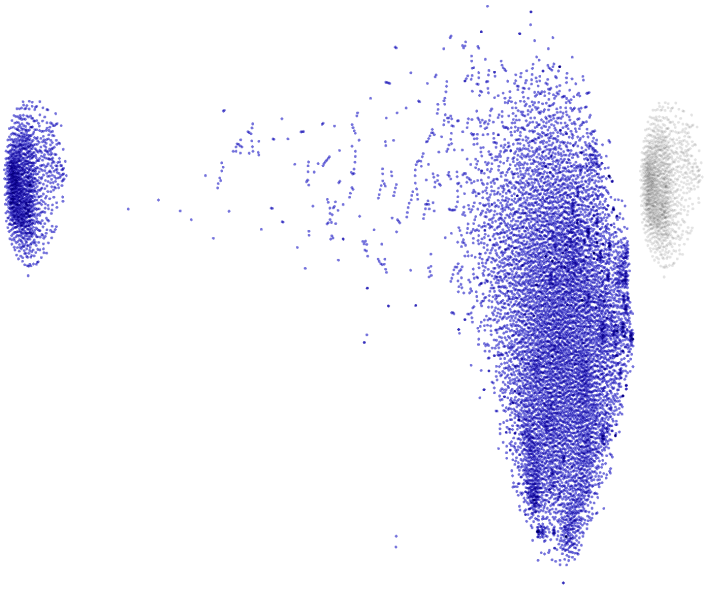
8.5 Trained Item Embeddings

As a final result of the trained embeddings, this section demonstrates how vectors representing items can be visualized. The fully trained item embeddings are plotted in a two-dimensional space using t-SNE — a method for exploring high-dimensional data (van der Maaten and Hinton, 2008). The goal is to reduce high-dimensional data into a two-dimensional space, illustrating similarities between embeddings by grouping items sharing features as discussed in section 7.1.0.1. We have been given access to item information beside the IDs for the AVM data, allowing us to exemplify and visualize item neighborhoods.[5]

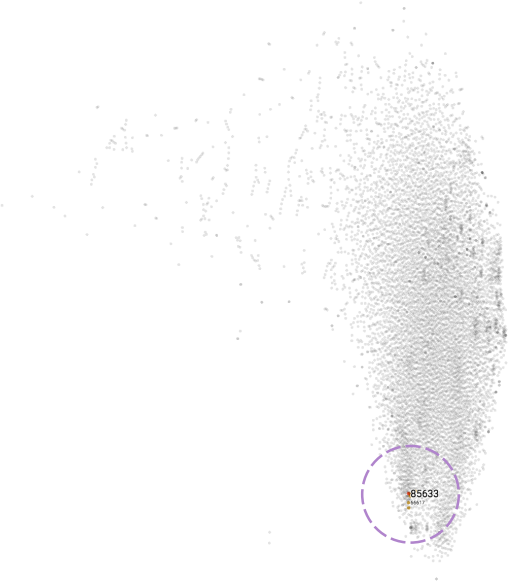
Figure 8.2 illustrates four plots made using t-SNE with a sample of 10,000 trained AVM item embeddings. Figure 8.2a shows the full plot without any highlighted items, whereas figures 8.2b, 8.2c and 8.2d show neighborhoods of similar items. From each neighborhood, we picked five items to check their similarity in terms of genre, artist and title. We found that each neighborhood share similarities — one neighborhood consists of children's books, one of Oldies (old music hits) and one of Elvis Presley vinyls.

Table 8.3 gives an overview of the similarities shared by items within neighborhoods. We observe that t-SNE successfully plots neighborhoods and in particular, we notice that all three samples of neighborhoods seem to have much in common. With this demonstration, we have successfully visualized and exemplified neighborhoods of similar items using the trained embeddings. It is essential to understand that similar embeddings are not final predictions of the presented recurrent neural network models. The presented networks will use the trained embeddings when propagating forward through the network to compute predictions, as covered in depth in section 4.2.2.

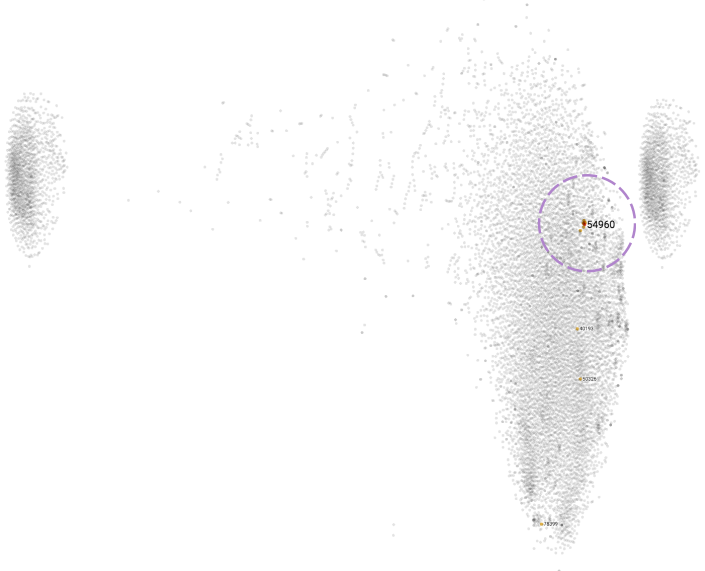
[5]When using t-SNE for plotting, one hyperparameter, perplexity, requires a value. According to Wattenberg et al. (2016) this value is typically set between 1 and 100. The perplexity of t-SNE is complex, and beyond the scope of this thesis. The value is a vague suggestion of the amount of neighbors each item embedding may have. We apply t-SNE using a perplexity of 10, as AVM remains a broad dataset considering unique items.



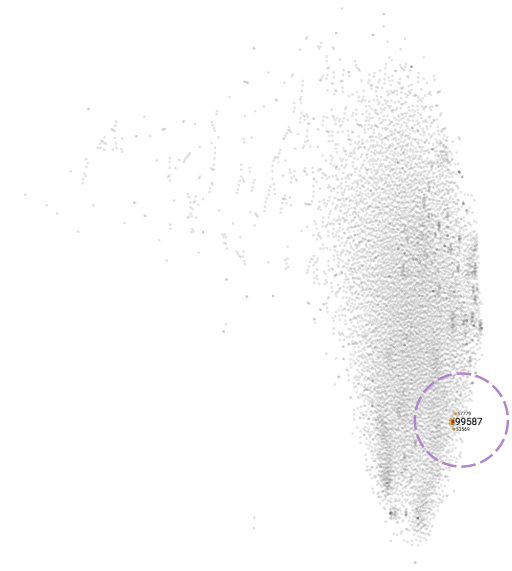
(a) No neighborhood illustrated



(b) Danish Children's Books



(c) Oldies



(d) Elvis Presley

Source: Own elaboration.

Figure 8.2: Plotting sample of 10,000 AVM item embeddings using t-SNE.

Table 8.3: Trained Item Embeddings — Sampled Neighborhoods

#	Artist	Title	Genre
8.2b	Marie Duedahl	*Lydret - En SMS fra Pi*	Children's Books
8.2b	JohannaLindbck	*Bare Sdan Lidt*	Books for Teenagers
8.2b	Ciaran Murtagh	*Finn Spencers Finntastiske [...]*	Children's Books
8.2b	Jim Hoejberg	*De Magiske Aeg*	
8.2b	Jim Hoejberg	*Ildens Vogter*	
8.2c	Brimstone Coven	*Black Magic*	Heavy Metal
8.2c	Alain Bashung	*Osez Bashung*	International
8.2c	Atlas Sound	*Logos Rock*	
8.2c	Dolly Parton	*9 to 5 & Odd Jobs*	Country
8.2c	Deep Purple	*Book of Taliesyn-col.vin*	Rock
8.2d	Elvis Presley	*That's the Way It is*	Rock n Roll
8.2d	Elvis Presley	*His Hand in Mine*	Rock & Roll
8.2d	Elvis Presley	*His Hand in Mine Gospel EP*	Rock & Roll
8.2d	Elvis Presley	*Blue Christmas - White Xmas [...]*	Rock & Roll
8.2d	Elvis Presley	*On Stage 1970 - Hq*	Rock & Roll

8.6 Summary

In this chapter we present final results computed upon a holdout testing sample. These results are used to compare performances of models and different data augmentation techniques. The data augmentation techniques did not improve RecSys' accuracy, but input dropout and reverse sequences positively affected AVM's accuracy. We showed that the lack of improvement on RecSys was due to its continuously changing trends. These temporal changes were accounted for during training of the model, by training from the oldest to the most recent events, which shows satisfactory results.

Looking into the results we showed supporting evidence that recurrent neural networks can overcome data sparsity and cold-start problems, whereas traditional models, such as the baseline kNN, suffer from these challenges. In addition, we argue that recurrent neural networks proved better at showing novel and diverse items. To ensure novelty and diversity in recommendations, we propose post-processes which includes removing items, which the specific user already saw, from the top 20 items predicted. The improved accuracy of the recurrent neural networks combined with more novelty and diversity in post-processes should increase chances of recommending items of serendipity.

Finally, we present a visualization of the fully trained item embeddings, illustrating neighborhoods of items. We briefly sample three neighborhoods, showing that embeddings have been trained with success.

CHAPTER 9

Discussions & Implications

The recurrent neural networks outperform the baseline model considerably on both datasets, but it may not be the case if the website is inherently different. Models presented were built upon elements that have "side effects". This chapter concentrates on discussing the benefits and drawbacks of the choices made during the research. Finally, the business implications of these models will be discussed to conclude on the value and practical use of the models.

9.1 Bias of Offline Testing

This applied offline evaluation method can be biased, as the e-commerce web-shops may already have implemented a recommendation engine. This recommender system may ultimately affect recommendations towards previously recommended items, as these are visited more regularly. Accounting for this bias is not possible as we do not have information about the type of event and whether the event came from a recommendation or not. However, for AVM, we know that no recommendation system was implemented at the start of this research. Hopefully, in the future, users will only choose to view recommended items, if the users finds the items interesting. Having users endorse recommendations that actually interests the users will gradually enforce the best recommendations for future users.

9.2 Local or Global Minima

As explained in section 7.2.1.2, we seek to gradually decrease the loss of the model until a minimum point is found. For each model, there will be multiple local minimums, but only one global minimum[1]. Unfortunately, we cannot know for sure whether the RecSys basic RNN or the AVM RNN RS finds the global minimum, of if the optimization algorithms

[1]The absolute lowest point.

get stuck in local minima. As extensive tuning of the model was done to reach the best results, we can assume that the results are close to a global minimum and thereby the lowest possible loss.

9.3 Data Processing and Augmentation

As covered in section 6.3, large parts of the AVM dataset were dropped due to a high bounce rate and about 75% of all sessions that had less than five events. To make a more robust and accurate model, we chose to remove these events, however, 85% of all items was removed from the AVM dataset, which limits the diversity and long tail items. Removing items with less than five events also removes events that occur in sessions that otherwise meet requirements for the quality of data. Sessions with long tail items may lose events in the sequence, that otherwise could contribute with information about user preferences. However, it was shown that removing this data reduces noise in the dataset and ultimately resulted in models with a much higher accuracy.

9.3.1 Rare Items

Session data holds timestamps of events. The total time of events was calculated by taking the difference between timestamps of current event and the following event of a session. However, since the last event of a session do not have a following event, the time cannot be derived. Session endings are caused by users leaving or if users are converted[2]. If users leave, we would want to remove the last item of the sessions, as this item could be of less interest. Whereas if a user ends up buying an item, the item would be considered highly relevant. Given the data at hand, we could not tell the difference and hence it was not possible to differentiate. Valuable data may have been dropped when applying the data augmentation technique removing items with only few events. This belief is supported by the fact that RNN BV performs slightly worse than the Basic RNN.

9.3.2 Computation of Data Augmentation

In order to decide if a model is superior with data augmentation, we take time of computation into account. Some techniques increased computation time by a factor of two, some decreased it. Session splitting, reverse sequence and briefly viewed items increases the data size and the possible increase in computation time must be weighed against improved accuracy. However, since training time was only doubled, we do not consider this critical when deciding on data augmentation techniques — especially if accuracy is improved.

[2]Transaction between webstore and customer.

9.4 The Costs & Benefits of Accuracy

The proposed recurrent neural networks outperform the baseline model and provide more relevant recommendations. Despite the large evidence that recommender systems increase the shopping experience, little is known about the specific business value of which e-commerce webstores gain from implementing recommender systems (Garfinkel et al., 2006). Ricci et al. (2011) found that conversion, cross-selling and customer loyalty increased with better recommendations. With this in mind, it remains difficult to link a value to any correct recommendation and a cost to any wrong recommendation.

9.4.1 Hypothetical Case Analysis

We state that a single conversion generates \$100.00 in profit and the conversion rate for an e-commerce webstore is around $\approx 1.5\%$ with a bounce rate of 60% (Compass, 2016). Every session would in average generate \$1.50 in profit. However, we know that sessions with only one event do not generate any profit and a session with multiple events must therefore generate a profit of $1.5/0.4 = \$3.75$. Converting bouncing users into staying at the webstore is potentially worth \$3.75.

About 35% of purchased items at Amazon was purchased due to recommendations (Company, 2013). Most of these can be assumed to be cross-selling while shopping. Amazon are pioneers within recommender systems; it is safe to assume that in general about 25% of sales should determine the maximum a recommender system can contribute in cross-selling — we call this measure γ. A study by Lee and Hosanagar (2016) concluded that using an item-based collaborative filtering technique improved the conversion rate, r, by 5.9%. Our models are 41.26% and 161.89% better in terms of accuracy, than that of the baseline model. If we assume a model will increase cross-selling by $\gamma = 25\%$, then the maximum expected increase in cross-selling, can determined by $(acc_{rnn} - acc_{base})\gamma$

Assuming that a conversion rate of $r = 5.9\%$ increases linearly and doubles when accuracy increases by 100%, we calculate an increase in conversion rate, Δr, by equation 9.1:

$$\Delta r = \left(\frac{acc_{rnn} - acc_{base}}{acc_{base}} + 1 \right) r \tag{9.1}$$

RecSys Expected increase in conversion rate by 8.33% and cross-selling by 5.25%, giving a total increase in revenue of **14.00%**

AVM Expected increase in conversion rate by 15.45% and cross-selling by 8.17%, giving a total increase in revenue of **37.39%**

To actually determine if a recommender system improves revenue, models should be implemented and tested using an online A/B testing. We argue that revenue can increase

when using recommender systems, but the cost of developing, implementing and maintaining such systems should be considered as well. Compared to a kNN model the cost of maintaining a recurrent neural network can be higher due to the required hardware configurations that allows for training neural networks in considerable time.

9.5 Black-box

When working with artificial neural networks and other model-based approaches, as described in section 3.1.2.2, the models easily become complex and hard to understand.

> *"Whenever you are modeling a problem, you are making a decision on the trade-off between model accuracy and model interpretation."*
>
> (Brownlee, 2014)

From a business perspective, better models increases revenue, brand-value and cross-selling. However, importance of being able to explain the behavior is for some businesses crucial. Therefore, the aspect of the model being a black-box should be weighed against other models that are more intuitive. In some applications, explanations of how and why are more important than the actual improvement in accuracy since businesses places its trust in a system which perhaps cannot be explained or interpreted. (Olden and Jackson, 2002)

9.6 Scalability

Using the proposed models as recommender systems in practice requires model scalability. The two datasets were utilized to test the models ability to scale between a high/low number of events and a high/low number of items. Only computational time was affected by the increase in a higher number of events or items. However, there are two sides of using this model in practice; training and inference. Inference is the models ability to generate recommendations of quality in a short period of time.

9.7 Conclusion

The objective of this thesis was to research how to improve user experience of new visitors by using recurrent neural networks and to determine if these outperform baseline models. Models is built upon data from two sources; RecSys — a large e-commerce webstore from Europe and AVM, a large webstore from Denmark. We cover the advantages of recurrent

neural networks over that of traditional recommender system techniques. In addition we sought to overcome traditional problems of recommender systems, such as data sparsity and cold-start problems. By reaching these objectives, the model can improve business values.

A wide variety of artificial neural networks exist, but this research utilizes recurrent neural networks, as these currently show state-of-the-art performances in other areas of research. Their main purpose is modeling sequential data, such as session data. To feed the recurrent neural networks with computational values, inspiration was drawn from natural language processing, helping us vectorizing the categorical item values into embeddings. Before modeling the data, data was structured, transformed and cleaned, leaving the AVM dataset significantly smaller.

As shown in chapter 8, recurrent neural networks have proven to outperform traditional models. The recurrent neural networks actually outperformed the baseline model, a traditional kNN, by 41.26% (RecSys) and 149.90% (AVM) in terms of accuracies. By using the full sequence of new users, the recommender systems can provide much more relevant recommendations. In addition, recurrent neural networks react to temporal changes and can quickly react to new user preferences. Especially in the RecSys dataset, we saw that temporal changes affect the model performance. Final accuracies was 71.55% for RecSys and 52.85% for AVM. The results also supports the belief that recurrent neural networks can overcome data sparsity problems, as AVM had a 45.58 times smaller events-per-item ratio compared to RecSys. These improvements conclude that artificial neural networks also tend to handle cold-start problems well, retaining a high accuracy.

The way the proposed models theoretically works will increase novelty, diversity, and serendipity more than in traditional models. Because the proposed models look at the users' path and predict the direction, the recommendations are more likely to become novel or diverse. We proposed using post processing on top predictions to boost novelty and diversity further. By improving novelty and diversity of recommendations, the model also increases the change of recommending items of serendipity to users.

To further improve the models, different data augmentation techniques were tested. Depending on the dataset, some techniques was found to improve accuracy slightly. None of the techniques improve accuracy on RecSys, but techniques including input dropout and reversed sequences had a positive effect on AVM, resulting in a model with 161.89% better accuracy compared to the baseline model. We can therefore conclude that data augmentation techniques also work with sequential categorical data, but the effect depend heavily on temporal changes and the sparsity of the dataset in question.

A hypothetical example shows that recommender systems based on recurrent neural networks can increase revenue by 14.00% (RecSys) or 38.39% (AVM). However, deciding between gain in revenue and understanding the model should be considered, as artificial

neural networks remain to be black-boxes.

chapter 10

Future Research

This research has brought forward a recurrent neural network that, with the use of data augmentation techniques, shows state-of-the-art results on the RecSys dataset. By implementing these models, one hypothesis would state that conversion rates and cross selling numbers would increase. As discussed in section 9, e-commerce stores, such as Amazon, increases their sales and revenue by modeling user behavior and utilizing recommendation systems based on these models.

In order to test the hypothesis of increased sales, the models would have to be implemented as actual recommendation systems. Due to limited time of research, any actual implementation has not been a possibility. Therefore, future work includes testing profitability of the developed models by comparing measures, such as conversion rate and size of basket, pre and post implementing the recommendation system. In addition, understanding the correlation between profit and increased accuracy is useful when considering the cost and benefits of developing models for recommendations. A deeper understanding hereof requires extended experiments.

Throughout this research, AVM has been supportive in generalizing the developed methods on to new use-case data, rather than the RecSys dataset. As results show, no model performance is equally good with two datasets. An interesting future topic could include many more e-commerce customer behavioral datasets with the purpose of proposing a finite set of tuning parameters, which tend to be significant for aligning the model architecture with new customer behavioral data. This could produce a basis model that, if tuned correctly, produces satisfactory results on new data.

To improve the performance of the model, researchers can look into the possibility of using known item features to include in the embedded item vectors. The embedding layer find features by looking at the user behavior. However, including content based features of the items may improve classification and hence the performance of the model.

The proposed models focused on improving accuracy for new customers by modeling session data. Combining the proposed model with another model using login data sources, such as gender, age, geographical location, purchase history, etc., could potentially im-

prove the customer experience of loyal customers further.

Bibliography

Deep image: Scaling up image recognition. *CoRR*, abs/1501.02876, 2015. URL `http://arxiv.org/abs/1501.02876`.

Charu C. Aggarwal. *Recommender Systems*, chapter 1, pages 1–28. Springer, 2016.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181. URL `http://dx.doi.org/10.1109/72.279181`.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35, 2013.

Erik Bernhardsson. Nearest neighbor methods and vector models part 1. `https://erikbern.com/2015/09/24/nearest-neighbor-methods-vector-models-part-1.html`, September 2015. Online; visited 24/05/2017.

Michael J. A. Berry and Gordon S. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Support*. Wiley, 1997.

Hossein Bidgoli. *The Internet Encyclopedia, G O (Volume 2)*. Wiley, 2003.

C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

John Blitzer, Kilian Q. Weinberger, Lawrence K. Saul, and Fernando C.N. Pereira. Hierarchical distributed representations for statistical language modeling. *University of Pennsylvania - ScholarlyCommons*, 2004.

Jason Brownlee. Model prediction accuracy versus interpretation in machine learning. `http://machinelearningmastery.com/model-prediction-versus-interpretation-in-machine-learning/`, August 2014. Online; visited 25/05/2017.

BusinessWeek. Interview with jeff bezos. `http://www.businessweek.com/ebiz/9903/316bezos.htm`, January 2003. Online; visited 16/03/1999.

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

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

McKinsey & Company. How retailers can keep up with consumers. `http://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers`, October 2013. Online; visited 20/05/2017.

Compass. Ecommerce conversion rates 2016 benchmarks. `https://blog.compass.co/ecommerce-conversion-rates-benchmarks-2016/`, September 2016. Online; visited 20/05/2017.

John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 4th Edition*. SAGE Publications, Inc, 2013.

Alexandre de Brebisson, tienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. Artificial neural networks applied to taxi destination prediction. *eprint arXiv:1508.00021*, 2015.

Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends in HumanComputer Interaction*, 4(2): 81–173, 2010.

Peter ffoulkes. The intelligent use of big data on an industrial scale, 2017.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *University of Cambridge*, 2016.

Robert Garfinkel, Ram Gopal, Bhavik Pathak, Rajkumar Venkatesan, and Fang Yin. Empirical analysis of the business value of recommender systems. *Available at SSRN: https://ssrn.com/abstract=958770*, 2006.

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

Google-Brain. Magenta. `https://magenta.tensorflow.org/blog/`, 2016. Online; visited 25/05/2017.

Egon Guba. *The Paradigm Dialog.* SAGE Publications, Inc, 1990.

Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn & Tensorflow.* O'Reilly Media Inc., 2017.

John Hearty. *Advanced Machine Learning with Python.* Packt Publishing, 2016.

Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM TRANSACTIONS ON INFORMATION SYSTEMS*, 22:5–53, 2004.

Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *International Conference on Learning Representations (ICLR) 2016*, 2016.

Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen.* PhD thesis, TU Munich, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117:500–544, 1952.

Stephen H. Kan. *Metrics and Models in Software Quality Engineering*, chapter 10, pages 302–305. Addison-Wesley Professional, 2003.

Mohammad Khoshneshin and W Nick Street. Collaborative filtering via euclidean embedding. *Proceedings of the fourth ACM conference on Recommender systems*, 2010.

Shah Khusro, Zafar Ali, and Irfan Ullah. Recommender systems: Issues, challenges, and research opportunities. *Springer Science + Business Media Singapore 2016*, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *eprint arXiv:1506.02557*, 2015.

Matevz Kunaver and Tomaz Pozrl. Diversity in recommender systems, a survey. *Knowledge-Based Systems*, 2017.

Dokyun Lee and Kartik Hosanagar. When do recommender systems work the best? the moderating effects of product attributes and consumer reviews on recommender performance. *International World Wide Web Conference Committee (IW3C2)*, 2016.

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. *ACM*, 2014.

Linyuan L, Mat Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. *Physics Reports*, 519(1):1 – 49, 2012.

Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L. Yuille. Explain images with multimodal recurrent neural networks. *Cornell University Library*, 2014.

Bernard Marr. A short history of deep learning – everyone should read. `https://www.forbes.com/sites/bernardmarr/2016/03/22/a-short-history-of-deep-learning-everyone-should-read/#583d2cb85561`, March 2016. Online; visited 25/05/2017.

Matthew R. McLaughlin and Jonathan L. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 329–336. ACM Press, 2004. URL `http://portal.acm.org/citation.cfm?id=1009050`.

Frank Meyer. *Recommender systems in industrial contexts*. PhD thesis, University of Grenoble, 2006.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Google inc. Mountain View*, 2013.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *eprint arXiv:1701.05369v2*, 2017.

Graham Neubig. *Machine Translation and Sequence to Sequence Models*, chapter 6, pages 33–44. Carnegie Mellon University, 2017.

Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. `http://neuralnetworksanddeeplearning.com/`.

Christopher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015. Online; visited 11/05/2017.

Julian D Olden and Donald A Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling*, 154(12):135 – 150, 2002.

Roger Parloff. Why deep learning is suddenly changing your life. `http://fortune.com/ai-artificial-intelligence-deep-machine-learning/`, 2015. Online; visited 16/05/2017.

Andrea Dal Pozzolo, Olivier Caelen, Gianluca Bontempi, and Universite Libre De Bruxelles. Comparison of balancing techniques for unbalanced datasets.

Sebastian Raschka. *Python Machine Learning*, chapter 12, page 344. Packt Publishing, 2015.

ACM RecSys. Recsys challenge 2015. `http://2015.recsyschallenge.com/index.html`, 2015. Online; visited 20/04/2017.

Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*, chapter 1, pages 1–35. Springer, 2011.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. pages 318–362, 1986. URL `http://dl.acm.org/citation.cfm?id=104279.104293`.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *GroupLens Research Group/Army HPC Research Center*, 2001.

J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 2001a.

J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Min. Knowl. Discov.*, 5(1-2):115–153, 2001b. ISSN 1384-5810. URL `http://dx.doi.org/10.1023/A:1009804230409`.

Herbert A. Simon. *The Sciences of the Artificial*, chapter 6, page 144. The MIT Press, 1996.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Hindawi Publishing Corporation*, 2009:1–19, 2009.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*, chapter 8, page 500. Pearson, 2005.

Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. *CoRR*, abs/1606.08117, 2016. URL `http://arxiv.org/abs/1606.08117`.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.

L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. 2008.

Martin Wattenberg, Fernanda Vigas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016. doi: 10.23915/distill.00002. URL `http://distill.pub/2016/misread-tsne`.

Shuxiang Xu and Ling Chen. A novel approach for determining the optimal number of hidden layer neurons for fnns and its application in data mining. *5th International Conference on Information Technology and Applications (ICITA 2008)*, 2008.

Xing Yi, Liangjie Hong, Erheng Zhong, Nathan Nan Liu, and Suju Rajan. Beyond clicks: Dwell time for personalization. *Proceedings of the 8th ACM Conference on Recommender systems*, 2014.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. *CoRR*, abs/1611.09100, 2016. URL `http://arxiv.org/abs/1611.09100`.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016. URL `http://arxiv.org/abs/1611.03530`.

Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005. doi: http://doi.acm.org/10.1145/1060745.1060754.