Designing a Recommender System based on Generative Adversarial Networks

Master's Thesis

Jonas Falkner
1717559

At the Department of Economics and Management
Institute of Information Systems and Marketing (IISM)
Information & Market Engineering

Reviewer:	Prof. Dr. rer. pol. Christof Weinhardt
Advisor:	Dominik Jung, M.Sc.
External advisor:	Dr. Robin Senge (inovex GmbH)

30. April 2018

Contents

List of Figures

List of Tables

List of Abbreviations

AF	Activation Function
ANN	Artificial Neural Network
BPR	Bayesian Personalized Ranking
BPTT	Backpropagation Through Time
BRNN	Bidirectional Recurrent Neural Network
CBF	Content-based Filtering
CF	Collaborative Filtering
CTR	Click-through Rate
DGP	Data Generating Process
DSRP	Design Science Research Process
DSS	Decision Support System
EBGAN	Energy-based Generative Adversarial Net
ETL	Expected Total Loss
EU	European Union
GAN	Generative Adversarial Net
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
IR	Information Retrieval
IRL	Inverse Reinforcement Learning
IS	Information System
LSTM	Long Short Term Memory
MC	Monte Carlo
MF	Matrix Factorization
ML	Machine Learning
MLP	Multilayer Perceptron
MRR	Mean Reciprocal Rank
MSE	Mean Squared Error
NIE	Normalized Inverse Entropy
NLL	Negative Log-Likelihood

NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RS	Recommender System
SeqGAN	Sequence Generative Adversarial Net
SGD	Stochastic Gradient Descent

1. Introduction

1.1 Motivation

Since the beginning of the World Wide Web an increasing amount of companies [1] [2] [3] successfully employs Decision Support Systems (DSS), precisely Recommender Systems (RS) to improve user satisfaction and to engage users in long-term relationships driving additional sales opportunities. This development has led to RS being an essential Information System (IS) providing strong competitive advantage [CWY04]; [Lü+12]. Therefore the field of RS has received a lot of scientific attention what resulted in enormous progress with respect to the employed methods and algorithms [Bob+13]. This development is the motivation for a work on advanced technologies for RS.

More precisely the employment of RS leads to several advantages on the customer side as well as on the business side. For the customer the advantage lies in an easy way to find new and interesting things. Therefore the RS supports the customer in effectively exploring the usually huge product space, narrowing down a particular set of choices while fostering the discovery of new, yet unseen products [Bob+13]. The business in turn can benefit from the conversion of mere visitors to profitable buyers since RS help them to find products they are willing to purchase. Furthermore RS yield many possibilities for the cross- and up-selling of products via automated and personalized recommendation targeting. An additional advantage is provided by improving customer experience and thereby driving customer loyalty. This is achieved by the RS inducing a *"value-added relationship"* between the customer and the site [SKR99]. Customers are much more likely to return to a site that offers a positive user experience by providing an easy to use interface and additional value-adding services. Finally the RS and its data can even be leveraged to predict future sales volume.

This thesis is written in cooperation with inovex[4]. The inovex GmbH is a German IT consultancy and project house with around 250 employees that is focusing on digital transformation. As a company providing consultancy services to small and large business partners alike one of its main contributions is the allocation of technical knowhow. This concept is very successful w.r.t. the deployment of RS since even large companies often do not have the knowhow and resources to design, develop and implement advanced recommendation technologies. This is due to the mathematical and computational complexity of RS as well as practical challenges involving the efficient processing of huge amounts of data, the requirement of recommendation in real-time and some RS specific problems which are explained in detail in 2.1.5. To this side the acquisition of new and innovative ideas and techniques plays an important role which motivates the cooperative work on this thesis.

The state-of-the-art in RS is indeed still mostly represented by *traditional* methods like matrix factorization models for Collaborative Filtering (CF) [KBV09]; [SLH14]. However

[1] www.amazon.com
[2] www.netflix.com
[3] www.spotify.com
[4] www.inovex.de/en/

these methods have problems to produce high quality recommendations when there is no or only little data about user-item interactions [SLH14] and when this lack cannot be compensated by side information either about the user or about the items [CCB11].

One of the most immense technical developments in computer science and adjacent domains in recent times is the rise of Artificial Neural Networks (ANN). Respective models show state-of-the-art performance on several Natural Language Processing (NLP) and image recognition tasks [Hin+12]; [KGB14]; [Rus+15]. This success led to the application of deep learning in other domains like RS.

Recent advances in ANN (especially Recurrent Neural Networks (RNN)) for RS are producing state-of-the-art results on session-only data [Hid+15]; [HK17]; [Qua+17]. These results were further improved by context and time sensitivity [SV17].

ANN have also disrupted the domain of image and text generation as well as machine translation with the deployment of Generative Adversarial Nets (GAN) [Goo+14]; [Im+16]; [Yan+17]. Furthermore recent improvements on the architecture and the training of GAN have rendered them applicable on a greater variety of problems, e.g sequential or discrete data [ZML16]; [Met+16]; [Yu+17].

As stated RNN work very well on sequential data while GAN show enormous potential concerning generation and transformation tasks and there are possible methods to apply them to tasks apart from the image and text domain. Therefore the driving idea of this thesis is to combine the capacity of these diverse technologies and employ them for the recommendation task on sequential data.

Beholding the recent attention to the respective research topics it is not surprising that there is already ongoing research on using GAN in conjunction with RNN to produce recommendations [Yoo+17]. However these are only emerging ideas that still present several problems and lack implementations and comparative studies to evaluate their real potential and value.

I intend to address these deficiencies in this master's thesis.

1.2 Problem Identification

Although a lot of advanced, successful RS technologies exist, in some restricted recommendation settings they nevertheless are confronted with problems that can lead to a severe drop in recommendation performance. One particular case to which this fact applies is session-based recommendation. The session-based recommendation setting is characterized by the absence of easily accessible user information preventing the creation of profiles or customer specific usage history [Hid+15]. Instead only the respective click-stream of a particular user session is available. This means the only information is which elements were consecutively clicked by a given user in the active session while the session can not be linked to previous sessions of the same user since the user is not completely identified.

This setting has recently become much more important because it presents a possible way helping to alleviate privacy issues. Privacy concerns about companies accumulating huge amounts of private user data and their usage patterns were already discussed years ago [LFR06]. However with the recent Facebook – Cambridge Analytica data scandal[5] that has received massive public attention, the debate has significantly gained momentum and shows that privacy is a very important subject to the overwhelming majority of people. The new General Data Protection Regulation[6] issued by the European Union (EU) in 2016 and applicable from 25 May 2018 is just another evidence for the prominence of personal data privacy. It is a new EU regulation to protect the personal data of EU citizens and empowers them with a new set of digital rights, e.g. the *right to erasure* or the *right to access* of personal data. Furthermore it extends the existing EU data protection laws to also apply to all foreign companies processing data of EU residents as well as harmonizing the different regulations for all states within the EU.

Furthermore for smaller e-commerce and news sites the session-based scenario arises almost naturally. Compared to globally acting multi-international enterprises the respective companies have the problem that depending on the marketed products consumers are not that frequently shopping in their e-commerce store and consequently the relinquishment of private data as well as the effort to create and maintain a profile often outweighs its utility. Nevertheless these companies are meanwhile confronted with a global market environment and therefore are forced to employ competitive RS technologies to sustain their market position [WR05]; [Dia+08]; [Hid+15].

Although there are already existing approaches to counter data privacy issues in RS [Aïm+08]; [Zha+10] differently, with view on the independent occurrence of session-based recommendation tasks I consider it crucial to design new RS technologies that are able to work in this particular setting.

1.3 Research Questions and Objectives

Current state-of-the-art recommendation results on session-only data are achieved by the RNN model developed by [Hid+15] and [HK17]. However I am interested in researching how to design a RS that employs RNN combined with adversarial training in the GAN framework. The motivation for this approach is based on the exceptional performance and results achieved by GAN in other domains. Therefore I want to see if the technique can be effectively applied for recommendation tasks. Furthermore I am going to investigate if this new type of RS is able to achieve similar or even better results than the existing model. From this general research goal I derived the following research questions to guide my investigation:

1. *How can a RS for session-based recommendation based on the GAN framework be designed?*
2. *How does such a RS perform w.r.t. recommendation relevance compared to the state-of-the-art in session-based recommendation?*

[5]www.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal
[6]www.eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679

1.4 Research Design

Since this thesis focuses on research applied to a practical problem, the *Design Science Research Process (DSRP)* proposed by Peffers et al. [Pef+06] that is depicted in figure 1.1 is well suited as orientation to guide the research.

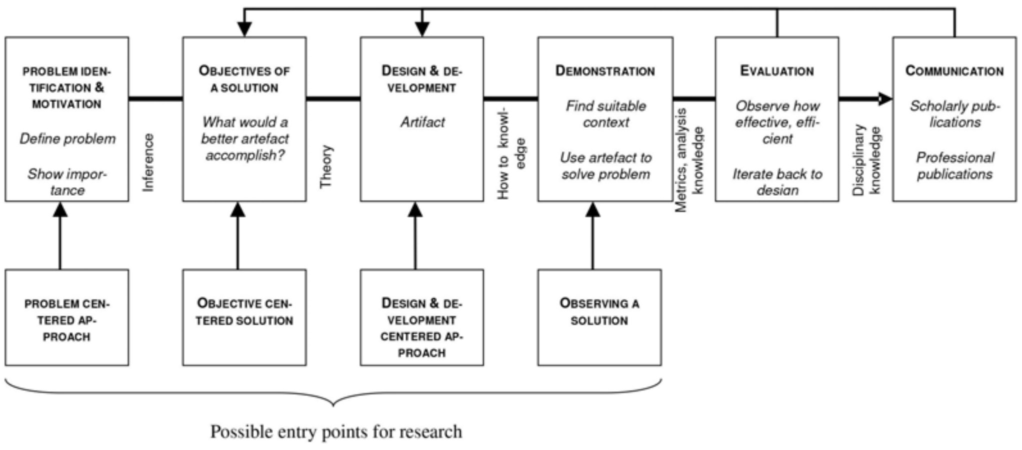
PROBLEM IDENTIFICATION & MOTIVATION
Define problem
Show importance
Inference
OBJECTIVES OF A SOLUTION
What would a better artefact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, analysis knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM CENTERED APPROACH
OBJECTIVE CENTERED SOLUTION
DESIGN & DEVELOPMENT CENTERED APPROACH
OBSERVING A SOLUTION
Possible entry points for research

Figure 1.1: Design Science Research Process model (Source: [Pef+06])

The process is made up of six consecutive steps that lead in a nominal sequence from problem identification over the development of a theoretical solution to its implementation and evaluation. The scientific contribution of this work especially focuses on the development, implementation and final evaluation of a model and its respective artifact (executable prototype of the model). Therefore the predominant weight lays on the steps three to five with a *design and development centered research approach.*

For the motivation of the thesis I want to point to the remarks in section 1.1 while the focus problem of the research is discussed in section 1.2. The general recommendation problem as well as more detailed application and implementation related problems are described in the sections 2.1 and 2.3. These problem statements lead to the definition of improvement objectives that should be fulfilled by the artifact and are explained in chapter 3 where details on the design and development can be found as well. The demonstration and evaluation are finally presented in chapter 4.

2. Theoretical Foundations

2.1 Recommender Systems

Generally speaking, RS are IS deployed to mitigate the problem of information overload that users find themselves confronted with in the modern world governed by a plenitude of easy available information and advertisement [Bob+13]. The general recommendation problem can be described as the problem of leveraging data on users and their preferences to predict their possible future likes and interests.

If we take this verbal problem more formally, let C be the set of all users and let S be the set of items. An item in this case can be every possible object that can be recommended, including e.g. music, movies and news as well as physical products and restaurants or physicians. Furthermore let u be an utility function modeling the utility that a user c obtains from the consumption of item s, that is given by $u : C \times S \to R$. Here R is a totally ordered set representing a rating that measures to which degree a user likes a particular item. Hence we are interested in finding for each user $c \in C$ an item $s' \in S$ that maximizes the respective utility [AT05]:

$$s'_c = \underset{s \in S}{argmax} \quad u(c, s), \quad \forall c \in C \tag{2.1}$$

The predominant proportion of the difficulty for this task belongs to the fact, that in general the utility is an arbitrary (complex) function that normally is only defined on a subset of $C \times S$. When speaking about ratings this means that for a particular user there are only some items available that the user has rated in the past. Furthermore the severity of the problem increases when considering that the rating information can be acquired *explicitly* (e.g. explicit user rating on a scale) as well as *implicitly* (observed user behavior e.g. item consumption) [Bob+13]. Therefore the need arises to extrapolate the known ratings to the whole space $C \times S$ to cover the unknown ratings. This is usually done by estimating the utility function u [AT05]. Based on the estimated utility function ratings for all items can be computed and the final recommendation is produced by either choosing the item with the highest rank or a list of the N highest ranking items (top-N recommendation). Depending on the specific application the recommended item might be already known to the user, however the most common case is to recommend items that the user has not seen yet.

In recent years lots of different approaches and models for the use in RS have been developed. The following paragraph presents and summarizes the main ideas and differences.

2.1.1 Content-based Filtering Models

First the arguably easiest method for recommendation is to recommend items to a user that are in some aspect similar to items he liked in the past. Therefore content-based filtering (CBF) includes attributes (meta data) of an item, e.g. to choose some films to be recommended to a user, one might not just pay attention to the respective genre or title, but also director, year of release and leading actors [Lü+12].

Additional attributes can further be computed by accessing the so called content of an item, that is composed of accompanying image, sound and video data that can be analyzed by modern *information retrieval* (IR) techniques [Lü+12].

In a consecutive step user profiles can be created by aggregating the attributes of the items a user liked in the past. Then the recommendation problem reduces to finding items with attributes that match a given user profile best. For this task several classification algorithms like *k-nearest neighbors*, *linear classifiers* and *probabilistic models* can be employed [Bob+13].

However the focus on attributes can lead to the problem of *overspecialization* for CBF, that is characterized by users only receiving recommendations for items that are very similar to items they liked before instead of recommendations for unknown but possibly more interesting items [Bob+13]. This problem is responsible for the development of hybrid methods that are presented in 2.1.3.

2.1.2 Collaborative-Filtering Models

The group of CF models is made up of two subgroups, the *memory-based* and *model-based* filtering approaches [AT05]; [Lü+12]; [Bob+13]; [SLH14]:

1. **Memory-based CF**
   Memory-based CF works directly on the so called user-item matrix containing the rating of user c for item s. In this approach the ratings are given directly or generated before the recommendation process, so the utility $u(c, s)$ is already tightly integrated in the ratings, which serve as "memory". Then similarity measures are employed to find suited items for a respective user. [Bob+13]. In this regard memory-based methods can once again be further divided into *user-based* and *item-based* approaches.

   The basic idea of user-based CF is to estimate the utility $u(c, s)$ based on the utilities $u(c_j, s)$ that other users $c_j \in C$ who are similar to user c assigned to item s [AT05]. This means we search for users that are similar to a particular user and aggregate their rating information to estimate his utility for each item. To find users that are similar to user c, popular similarity metrics are the *Pearson Correlation Coefficient* and the *Cosine Similarity.*

   In contrast the respective recommendation problem for item-based CF is to estimate the utility $u(c, s)$ based on the utilities $u(c, s_i)$ that have been assigned to items $s_i \in S$ by the user c in the past and are similar to item s [AT05]. To find items similar to the items that a user has already rated in the past the same aforementioned similarity metrics can be employed.

2. **Model-based CF**
   Model-based CF methods use the user-item matrix as immediate input for a model that is trained on top of it. This model then directly generates recommendations

for a user by employing the estimated utility function $u(c, s)$ learned by the model. Popular models include *Bayesian Networks*, *Mixture Models*, *Neural Networks* (see 2.3) and *Matrix Factorization* (MF) approaches [Bis16]; [Bob+13]; [SLH14].

2.1.3 Hybrid Models

Hybrid approaches use combinations of the aforementioned models to estimate the utility of a user. This structure allows for much more advanced and complex models that are able to adapt to varying scenarios and domains as well as to mitigate the accompanied problems presented in 2.1.5 much better than pure content-based or CF models.

There are several possible ways to combine models with different characteristics into a hybrid model. One popular approach is to integrate content information in CF by leveraging graph representations or clustering methods that combine the collaborative data of the user-item matrix with content information [PH15]; [Bob+13]. Another approach would be to build distinct CBF and CF models and then to train an additional model on top to combine the respective predictions, what is a common technique in ensemble learning and model averaging [AT05].

2.1.4 Neural Networks for Recommendation

In the field of general RS the state-of-the-art is mostly represented by CF approaches and ANN are usually only employed in advanced hybrid models [WWY15] or for highly specialized recommendation problems. In particular RNN had a substantial impact on RS working on sequential data (user sessions and click-streams) compared to the user-item data in CF. Here user profiles are often non-existent what prevents the use of MF models which work based on latent vectors for users and items and became well-known by winning the infamous Netflix contest [KBV09]. The groundwork for RNN-based recommendation was laid by Hidasi et al. [Hid+15] who achieved surprising results in the session recommendation task compared to traditional methods. Their approach received enormous attention from the scientific research community what led to rapid further development and model refinement [Hid+16]; [TXL16]; [HK17]; [Qua+17]; [SV17]

2.1.5 Common Challenges

In case a recommender model is deployed to the domain of the real world, it is incorporated by a RS and is faced with additional application specific problems that need to be addressed. These Problems include but are not limited to [Lü+12]:

1. **Relevance**
   The recommendations have to be relevant to the user. Although there is a manifold of methods to solve the recommendation task, there is still room for the improvement of relevance and accuracy [LDP10]; [Ji+16]; [Dai+16]; [SV17].

2. **Scalability**
   The RS architecture needs to be scalable to huge sets of users and items w.r.t. memory and computational complexity at runtime.

3. **Cold-start**
   New users and items for which there is no or only little information available yet (in my case user sessions that are to short for usual approaches) have to be integrated in the RS.

Furthermore there are additional problems caused by the structure of sequential data and the domains in which this sort of data is common:

4. **Absence of user profiles**
   Users are not logged-in or not being tracked when interacting with the RS and consequently user profiles and the user-item matrix are inexistent [Hid+15].
5. **Timeliness**
   The utility of a certain item strongly depends on the time it is recommended, e.g. news is mostly up-to-date (and therefore valuable) only for a restricted period of time [Ji+16]; [Dai+16].
6. **Thematic drift**
   The interests of users change dynamically over time what makes predictions difficult and reduces the power of pure item-to-item methods [LDP10].

2.2 Machine Learning Fundamentals

The area of machine learning (ML) has seen tremendous interest in recent years. Last but not least thanks to the Internet and a globally connected world huge amounts of data are produced each second [Par11]. That is the reason why an increasing number of people and companies want to make sense from that data to fuel their private and commercial interests. The way to do that is mostly represented by a plenitude of machine learning algorithms from simple regression models to deep convolutional ANN. However there are some basic concepts that remain the same over the overwhelming majority of models. These will be introduced in the following paragraphs.

2.2.1 The General Learning Task

A general definition of the learning task for a machine or program is stated by Mitchell in his 1997 handbook on machine learning [Mit+97]:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

The class of tasks T represents the hypothesis space $\mathcal{H}^\alpha$ that is defined by $\{h_\alpha : \alpha \in A\}$, where A is the parameter space defining the particular hypotheses. The learner is entrusted to find the optimal model parameters α_{opt} by learning from a limited amount of known examples. In other words the task is to estimate a model $h_{\alpha_{opt}}(x) = y$ that explains the observed examples (x, y) which are generated by an unknown and unobservable underlying process. Here x is the input sample and y represents the label of the sample. In applied machine learning jargon these examples are also called the *training data*, because they constitute the data that is available to train the model.

Within the process three important problems can be characterized:

1. **Statistical Problem**
   Compared to the available amount of examples the dimensionality of A is to high. This results in several hypotheses being equally suited for the task at hand, so no definite decision about the optimal model $h_{\alpha_{opt}}$ can be derived.
2. **Complexity Problem**
   The complexity of the problem might prevent the learning algorithm from arriving at an optimal solution, e.g. when heuristics have to be employed since there is no known analytical method to restrain the complexity of the given problem. In this case the found solution might be suboptimal.
3. **Representation Problem**
   There might be no suited hypothesis in the examined space of hypotheses, therefore the required goodness of approximation cannot be achieved.

A more detailed discussion can be found in [Vap99].

2.2.2 The Objective Function

The objective function defines the general objective of our learning task. It guides the training process towards a specific direction by defining the absolute goal as well as by imposing restrictions that decrease the dimensionality of the possibly unlimited space of hypotheses. It usually includes a loss function.

2.2.3 The Loss Function

In order to find a optimal solution we need a suited performance measure P to determine the best model $h_{\alpha_{opt}}$ for the task T. This performance measure is often represented by a so called loss function that is specific to T [GBC16]. We denote the loss function by

$$\mathcal{L}(h_\alpha) = \mathcal{L}(h_\alpha(x), y) \tag{2.2}$$

Since our training data is limited, it is impossible to calculate the real error of the learned model compared to the underlying true model. However loss functions in combination with other methods represent a way to estimate this error.

As stated before, the loss function has to be chosen w.r.t to the learning task T at hand. The two learning tasks most common are *regression* that for instance is concerned with fitting a curve to a given data manifold and *classification* which seeks to assign the data to one of k categories or classes.

The average expected loss is given by

$$E[\mathcal{L}] = \iint \mathcal{L}(h_\alpha(x), y) p(x, y) \; dxdy, \tag{2.3}$$

where $p(x, y)$ is the probability density function of the data generating process (DGP).

For most regression problems the *squared loss* which punishes large absolute differences quadratically is a common choice [Bis16]:

$$\mathcal{L}(h_\alpha) = (h_\alpha(x) - y)^2. \tag{2.4}$$

While this works quite well for regression problems, in a classification setting we usually calculate the loss with respect to class memberships. Therefore we need other measures that compensate for that particular requirement. For a binary classification problem, where $k = 2$ the *logistic loss* is a plausible candidate [Ros+04]:

$$\mathcal{L}(h_\alpha) = \frac{1}{\ln 2} \ln(1 + e^{-y\ h_\alpha(x)}), \tag{2.5}$$

where the label $y \in \{-1, 1\}$ denotes one of the two classes.

Another possible measure is the (binary) *cross-entropy* that compares a distribution q (our model) to a fixed reference distribution p (DGP):

$$\mathcal{L}(h_\alpha) = -y \log h_\alpha(x) - (1 - y) \log(1 - h_\alpha(x)), \tag{2.6}$$

where in this case $y \in \{0, 1\}$.

Finally in the multi-class case with $k > 2$ often the *Negative Log-Likelihood* (NLL) loss (corresponding to the cross-entropy for a multinomial distribution) is employed [Bis16]:

$$\mathcal{L}(h_\alpha) = -y \log h_\alpha(x), \tag{2.7}$$

where $y \in [0, 1]$ and $h_\alpha(x) \in [0, 1]$ now are the respective class probabilities.

2.2.4 The Bias-Variance Trade-off

A common way to evaluate the performance of a model empirically is to use a so called *hold-out* dataset. Therefore the available data is split into a *training set* that is used to train the model and a separate *test set* that is held-out, meaning not used for training. The procedure is to train the model on the training set and then to evaluate its performance by calculating predictions on the test set that hasn't been seen by the model yet, but whose labels are known. The motivation for this procedure is the fact that we don't want our model to produce good results for data it has already seen, but for new real data whose labels are not known. The ultimate goal for the model is to achieve a good generalization.

If the model includes hyperparameters that are not learned in the process one might even split the training set again, into a real training and a *validation set* that is used to infer a adequate set of hyperparameters. In case that the available data is quite limited one might consider using advanced methods like cross-validation or bootstrapping [Mit+97].

The assumption that justifies such a split of the original data to draw conclusions on the expected performance is the so called i.i.d. characteristic that states that we assume all examples in the data to be independent of each other and that two arbitrary sub sets of the data are identically distributed [GBC16].

Taking a closer look at the squared loss function of regression, one can show that it can be brought into a form that is made up of a squared expectation value that depends on the model h_α and an independent error term [Bis16]:

$$
\begin{aligned}
E[\mathcal{L}] = \int \{h_\alpha(x) - E[y|x]\}^2 p(x)\ dx + \{E[y|x] - y\}^2 p(x, y)\ dx \\
= E[y - h_\alpha(x)]^2 + Var(\epsilon) \quad (2.8)
\end{aligned}
$$

The first term represents the reducible proportion of the error. It is minimized by the most appropriate model. The second term in contrast consists basically of statistical noise that is independent of the model and therefore cannot be reduced.

This characteristic of the loss leads to a very essential problem of ML: The *bias-variance trade-off*. If we think about the problem of fitting a curve to a point cloud in the two dimensional space, we can either use a straight linear regression line or polynomials of increasingly high degree. The problem is illustrated in figure 2.1. On the left we see the data that is sampled from the true underlying function shown in black.

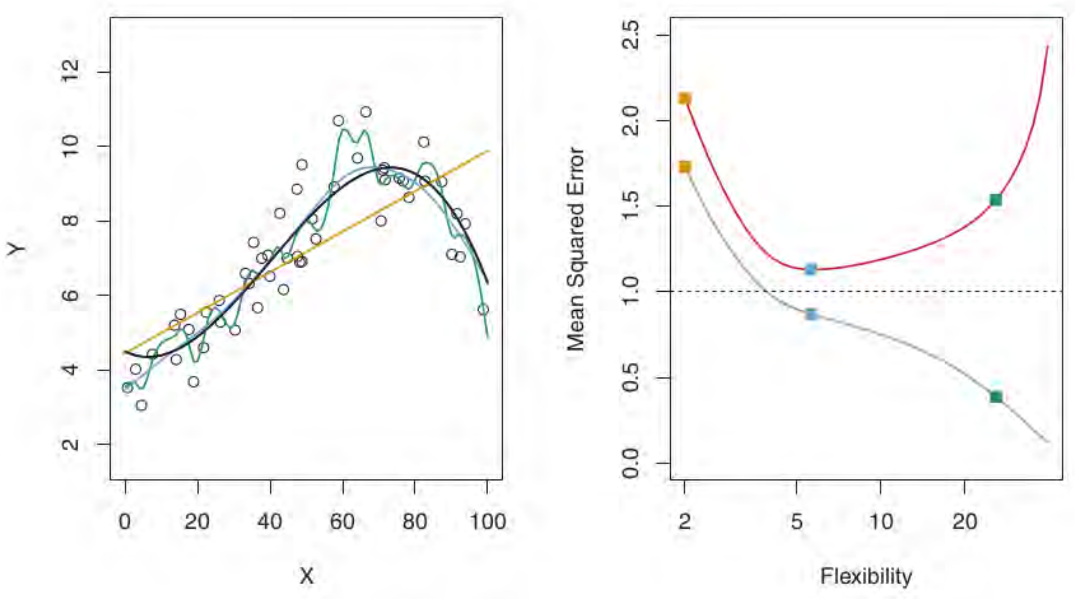

Figure 2.1: Illustration of the bias-variance trade-off in a exemplary curve fitting problem (source: [Jam+13], p.34).

The simple linear model (orange) is quite restrictive and fails to capture part of the variation of the data. On the other hand its shape usually does not change by much if one of the data points is moved by a small amount. It is described by having high bias but low variance. The polynomial of high degree (green) on the other hand will fit the data points very well and accordingly has low bias, but will change significantly if one of the points is moved and therefore has high variance. The best fit is achieved by a model of moderate flexibility (blue). However the inherent problem is, that we cannot reduce both the bias and the variance at the same time. This behavior is called the bias-variance trade-off. If we plot the respective error of the model on the training set (gray line) and the test set (red line) we can see that for some area of the loss curve the error decreases and afterwards starts to increase again. The result is the characteristic *U-shape* of the test error curve that can be seen on the right hand side of figure 2.1.

We take this observation as motivation to have another look at the expected loss of our model. For a given value x_0 it can be decomposed into the sum of three fundamental quantities [Bis16]; [Jam+13]:

$$E[\mathcal{L}(h_\alpha(x_0))] = Var(h_\alpha(x_0)) + [Bias(h_\alpha(x_0))]^2 + Var(\epsilon) \tag{2.9}$$

Hence the U-shape of the error originates from two essential parts of the loss that are responsible for the respective error, namely the variance and the bias of the model. Obviously there is an area within that the bias reduction by a increasingly flexible model is of higher absolute value than the negative contribution of the increasing variance.

The behavior of a very flexible model to follow the shape of the point cloud very closely and consequently the danger of partly modeling the inherent noise (irreducible error) instead of the true underlying distribution of the DGP is called *overfitting*. This behavior reduces the ability of the model to generalize beyond the training data and the test error increases accordingly. Therefore this characteristic is an undesirable property and precautions need to be put in place to prevent a model from overfitting. Respective techniques for ANN are explained in 2.3.1.5.

The tendency of a model to overfit depends on its *capacity*. A popular measure for the capacity of a hypothesis space is the so called *Vapnik-Chervonenkis dimension* (VC-dimension). Basically it is an approximate measure for the maximal number of arbitrarily set labels for which the data points of a set S can be separated from $\mathcal{H}^\alpha$ [Vap99].

2.2.5 No Free Lunch in Machine Learning

The infamous *no-free-lunch theorem* [WM97] states that for certain problems given by a particular objective function, or rather a particular probability distribution, all possible models produce identically distributed results. This statement can be loosely interpreted as there is no ”one size fits all” approach to construct a model for learning tasks. Consequently there is no universally optimal approach to solve a ML problem. However an algorithm might outperform some other when altered to solve a specific and restricted problem class, accepting that its results on other problems might decline in the process.

2.2.6 The Curse of Dimensionality

Another important notion in ML is the so called *curse of dimensionality* [Bel61]. It describes the fact that when increasing the feature space, the required amount of training data to ensure that each combination of values is represented by a sufficient number of samples grows approximately exponentially. Moreover the same is true for the parameter space when we try to fit a model for a high dimensional problem. This certainly poses a serious problem for general ML approaches, however effective algorithms for the application in high dimensional space have been developed. Respective models make use of two characteristics inherent to most real data [Bis16]:

1. Most of the variation in real data that is no statistical noise is confined to a subspace of lower dimension, what e.g. is leveraged by dimensionality reduction methods like principal component analysis [Jol02].
2. Typically real data exhibits some local smoothness properties that guarantee that some small changes to the variables in the input dimension will lead to some small changes in the output. Then this characteristic can be exploited to infer predictions for new input values.

2.3 Artificial Neural Networks

Substantially an ANN can be understood as a general function approximator. The approximated function can have arbitrary form but usually is a classification function $y = f(x)$ that maps the input vector x to a categorical vector $y \in Y$ where Y is the set of classes, with $|Y| \geq 2$. The respective mapping is governed by a set of learnable parameters θ that are dynamically adapted in the learning process to produce the best approximation $\hat{y} = f^*(x; \theta)$ [GBC16].

In the next paragraph the most basic form of an ANN, the feed forward neural network is described. Moreover the training process for the parameters is explained and some aspects of the respective architecture design decisions are clarified. In the following chapters advanced model structures like recurrent neural networks and generative adversarial nets are introduced.

2.3.1 Feed forward Neural Networks

The simplest form of an ANN is the feed forward neural network, sometimes also called *Multilayer Perceptron* (MLP). However, this designation is a bit unfortunate because a feed forward neural network is rather composed of several layers of logistic regression models [Bis16] with continuous non-linearities compared to the discontinuous case of a Perceptron [Ros58].

The name ”feed forward” is based on the notion of information flowing through the network from an *input layer* that receives the input x, over possibly multiple so called *hidden layers* to a final *output layer* producing the output $\hat{y}$. Each layer is connected only to the preceding layer that feeds information into it and to the subsequent layer. There are no links connecting non-adjacent layers [GBC16].

Recently the notion of *deep artificial neural networks* or *deep learning* has appeared. Basically it describes ANN with several hidden layers however it is an ongoing argument from how many layers an ANN should be called 'deep'. For simplicity in this thesis all ANN with at least two ore more hidden layers are considered deep.

Each layer consists of one or more so called *neurons*. Neurons are the elemental building block of an ANN and basically define a mapping, usually from an vector-valued input to a scalar output space:

$$a_j = y(x, w) = \phi(\sum_i w_{ij} x_i + w_{j0}) \tag{2.10}$$

where $\sum_i w_i x_i$ is the weighted sum over the input to neuron j and w_{j0} is an additional bias term. In vectorized form we often concatenate the input with a +1 at the first position to absorb the bias in the weight vector. The function ϕ is called an *activation function* (AF). The activation function introduces a non-linearity to the neuron and thereby increases the capacity of the model [Bis16]. Common activation functions are presented in 2.3.1.3. The structure of a basic neuron is depicted in figure 2.2.

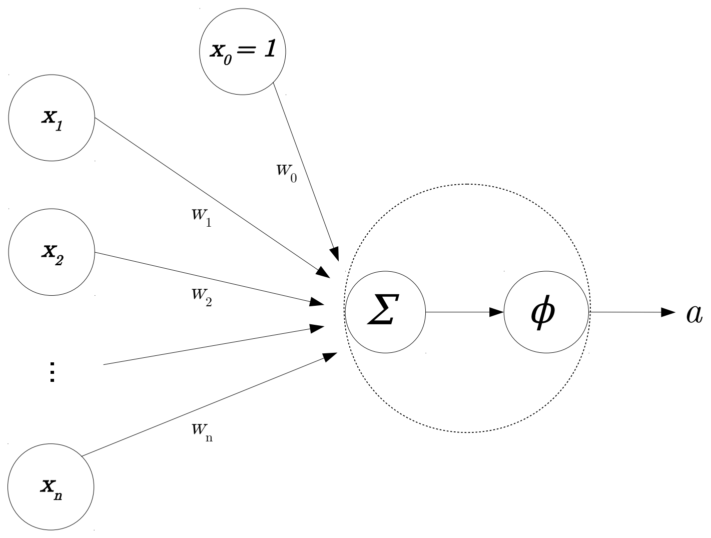

Figure 2.2: Basic structure of an artificial neuron.

In an ANN several of these neurons work in concert to form a network that transforms the input x to the output y. Each layer normally consists of multiple neurons that all receive their input from the preceding layer. Since the intermediate layers are connected to the input and output only by the most outer layers, they are called "hidden".

Each neuron is basically made up of a series of functional transformations, therefore in the following a neuron will be denoted by $h_j^{(l)}$ where the subscript describes the j-th neuron in a layer and the superscript indicates the l-th layer of the network to unclatter notation. The input layer is denoted by $h^{(0)}$. Along these lines we characterize the dimension of a network by its depth L that determines the number of consecutive layers and the width

N_l that describes the number of neurons in each layer l.

2.3.1.1 The Forward Pass

In the forward pass the input x provides the initial information for the network. It is received by the input layer $h^{(0)}$ and then propagated through the hidden layers of the network by consecutively using equation 2.10 on the output of the preceding layer:

$$h_j^{(l)} = \phi(\sum_i w_{ij}^{(l)} h_j^{(l-1)} + w_{j0}^{(l)}) \tag{2.11}$$

what in vectorized form can be written as

$$h^{(l)} = \phi(w^{(l)} h^{(l-1)}) \tag{2.12}$$

Ultimately the information passes the output activation of the output layer and results in a final prediction value $\hat{y}$. Then a loss function is employed to measure the difference between the network output $\hat{y}$ and the real data y, producing the loss:

$$Loss = \mathcal{L}(\hat{y}, y) = \mathcal{L}(h^{(L)}, y) \tag{2.13}$$

Two important assumptions in this process are that the loss can be calculated by averaging over the individual loss functions of each output neuron and to be able to use gradient descent that it is at least once differentiable w.r.t. its input [Bis16].

2.3.1.2 The Backpropagation Algorithm

After calculating the loss, we essentially want to know how we need to adapt the weights of the network to better approximate the unknown function f. The backpropagation algorithm [RHW86], sometimes also just called ”backprop”, is a method to do that.

Basically the idea is to compute the partial derivative of the loss function w.r.t. the weights and the biases of the network. We can think of it as trying to change the weights by a small amount $w = w + \delta w$ and observing a change of the loss function given by $\delta\mathcal{L} \simeq \delta w^T \nabla\mathcal{L}$, where $\nabla\mathcal{L}$ is the gradient of the loss function that points into the direction where the loss function has the highest rate of increase [Bis16]. This means that to decrease the loss we want to make a small step in the negative gradient direction $-\nabla\mathcal{L}$.

So for each individual weight, we need to calculate the partial derivatives $\partial\mathcal{L}/\partial w_{ij}$. The backpropagation algorithm does that efficiently by using the chain rule of calculus [GBC16].

Applied to the partial derivatives this rule yields:

$$\frac{\partial\mathcal{L}}{\partial w_{ij}} = \frac{\partial\mathcal{L}}{\partial h_j}\frac{\partial h_j}{\partial w_{ij}} \tag{2.14}$$

In vector notation this results in a matrix product of the Jacobian with a vector gradient:

$$\nabla_w \mathcal{L} = \left(\frac{\partial h}{\partial w}\right)^T \nabla_h \mathcal{L} \tag{2.15}$$

Consequently the backpropagation algorithm performs such a Jacobian-gradient product for the weights of each layer. The resulting gradient direction is then used with a step size in an optimization algorithm like gradient descent and its variants discussed in 2.3.1.6. This particular technique is one of the main features of modern ANN and altered versions are used in advanced networks like the RNN described in 2.3.2 as well.

2.3.1.3 Activation Functions

As already stated earlier in this chapter, an activation function ϕ produces a non-linear transformation of the input. A manifold of AF has been proposed for the use in ANN and all of them have been shown to have some advantages and issues [GB10].

The early Perceptron model developed by Rosenblatt [Ros58] employs a *Heavyside step function*, that in its discrete form is given by:

$$\phi(x) = \begin{cases} 0 & for \quad x < 0 \\ 1 & for \quad x \geq 0 \end{cases} \tag{2.16}$$

For the simple case of a Perceptron this AF is sufficient, however more complex architectures including backpropagation suffer from its discontinuous character and the undefined derivative at $x = 0$.

Another popular legacy AF is the *logistic function* also called *sigmoid* [GBC16]:

$$\phi(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

The sigmoid is a simple function mapping from $\mathbb{R}$ to the closed interval of $(0, 1)$ but has a problem with *neuron saturation.* This describes the phenomenon of the output of the sigmoid becoming approximately zero for very small or very large input values, since the slope becomes very "flat" at the edges [GB10]. Therefore it isn't used as AF for practical problems anymore but rather the *hyperbolic tangent* that does not suffer as much from saturation:

$$\phi(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.18}$$

Another AF that recently is recommended as the "gold standard" for feed forward neural networks is the *Rectified Linear Unit* (ReLU) [NH10]:

$$\phi(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases} \tag{2.19}$$

Moreover improved results especially for deep convolutional neural networks are reported for an altered version of the ReLU, the *Leaky Rectified Linear Unit* [MHN13], that treats the slope parameter λ as a hyperparameter:

$$\phi(x) = \begin{cases} \lambda x \quad for \quad x < 0 \\ x \quad for \quad x \geq 0 \end{cases} \tag{2.20}$$

2.3.1.4 Output Functions

Compared to the AF of hidden neurons, the activation function ϕ^L for the output layer needs to satisfy different requirements. It doesn't have as much a problem with saturation but is needed to transform the final activation into the label space of y. For a classical regression problem or respectively a Gaussian distribution of the output, this label space is the standard $\mathbb{R}$ or $\mathbb{R}^n$ that usually doesn't depend an additional transformation. However the same is not true for binary or multi-class problems that require ϕ to be a *discriminant function* mapping the input into a discrete space $\mathbb{C}_k$ [Bis16]. To this end the sigmoid 2.17 is essential in directly assigning a class probability for the binary case. The same is possible for the multi-class case by introducing the so called *softmax function*, that basically is the continuous approximation to the maximum function and creates a mapping $\mathbb{R}^k \to [0,1]$ from a k-dimensional vector space into the real valued closed interval of $[0,1]$ that therefore can be interpreted again as probability vector over the k classes:

$$\phi(x) = softmax(x) = \frac{e^x}{\sum_i e^{x_i}} \tag{2.21}$$

2.3.1.5 Regularization

In chapter 2.2.4 we already perceived the notion of overfitting. Since ANN are models of increasingly high capacity the deeper they get, arrangements to prevent overfitting are inevitable. Fortunately several methods for the regularization of ANN have been developed:

- **Weight norm**
  The weight norm usually introduces an additional regularization term to the objective function to limit the capacity of the model by explicitly forcing the norm of the model parameters to be small:

  $$J(\theta) = \mathcal{L}(f_\theta(x), y) + \lambda\Omega(\theta) \tag{2.22}$$

  with the hyperparameter $\lambda \in [0, \infty)$ and a norm Ω.

  In case of neural networks the regularized parameters θ normally only consist of the weights without the particular biases of each layer. Therefore in the context of the weight norm for ANN θ is represented by the weight matrix w. A popular choice for the norm Ω is the euclidean or L^2 norm that punishes large parameter values quadratically and is also known as *Tikhonov regularization* [Tik43]:

$$J(w) = \mathcal{L}(f_w(x), y) + \lambda \frac{1}{2} w^T w \tag{2.23}$$

- **Dropout**
  A regularization method that is exclusive to ANN is the so called dropout [Sri+14]. It was designed to improve the generalization capability of ANN by randomly setting the activation of individual neurons in each training iteration to zero. The probability with which the "dropping out" of neurons happens is specified by a dropout probability parameter p. This procedure forces the network to create structures that are partly interchangeable and that do not depend to much on single neurons that might be dropped. Adapting to this method essentially leads to a more general structure and thereby reduces overfitting to the training set. Another explanation for this behavior is that the dropout procedure represents an approximation to a bagged ensemble of neural networks that is known to reduce the variance of the final model while leaving the magnitude of the bias unchanged [NIH97]. For the prediction on the test set the dropout is finally omitted to exploit the full potential of the network.
- **Early stopping**
  Although early stopping is basically no mathematical but a rather empirical approach to regularization, it has its own justification [YRC07]. The early stopping rule states that one should stop the training of the network if for a certain number of epochs no significant decrease of the error rate on the test or rather validation set can be achieved. A popular interpretation is e.g. the *one-in-ten* rule that recommends to stop if there was no significant improvement within 10 epochs. All of this might look a bit arbitrary, since the general number of epochs can vary greatly depending on the training data and network architecture as well as there is much freedom to define what a "significant" improvement is. Nevertheless the procedure makes sense if we recall that overfitting does not only depend on the general capacity of the model but also the length of training or rather on how often the model learns from the same available training data that doesn't change between the respective epochs.

2.3.1.6 Optimization Methods

In this subsection we will discuss several methods to optimize our model. In the particular case of ANN these algorithms are nearly exclusively so called *descent methods*. Basically such methods try to minimize a given function by consecutively applying the following general update rule [BV04]:

$$x^{(t+1)} = x^{(t)} + \eta \Delta x^{(t)}, \tag{2.24}$$

where η is the *step size* also called *learning rate* in ANN jargon and Δx is the *search direction*. The created sequence x^k with $k = 1, ..., k_{opt}$ is called the *minimizing sequence*.

Generally speaking we are searching a suited direction to make a step in order to descent the slope of the given function and finally reach its minimum. Formally we can describe this behavior by requiring the method to satisfy

$$f(x^{(t+1)}) < f(x^{(t)}), \tag{2.25}$$

for all x that are not optimal.

The algorithms presented in the following essentially differ only in the way they choose a learning rate η and the search direction Δx.

Gradient Descent

The basic idea of the standard gradient descent method is to choose the negative gradient as search direction [BV04]. This choice makes inherently sense since the gradient points into the direction where the function has the highest rate of increase, so we hope to select an acceptably good search direction by going in the opposite direction. Our update rule therefore becomes:

$$x^{(t+1)} = x^{(t)} - \eta \nabla f(x). \tag{2.26}$$

In practice there are three possible variants of the basic gradient descent that are used in ML models. They differ in which part of the training data they use to calculate the negative gradient direction. The most obvious approach called *batch gradient descent* is to calculate the gradient over the whole dataset, then do the according optimization step and repeat the procedure until a stopping criterion is satisfied. This is possibly the most accurate way of doing the gradient descent, however for large datasets this method is slow and consequently not practical.

The other extreme is found in the *Stochastic Gradient Descent* (SGD) that instead of the whole dataset uses only one individual training example for the gradient computation. However this approach is only a vague approximation of the real gradient and comes with a high variance for the updates, that leads to a "zig-zagging" behavior that is illustrated in figure 2.3. The zig-zagging is also apparent for the standard batch gradient descent but is much severe in case of the SGD what leads to slow convergence.

Due to the disadvantages of the previous methods the approach that is mostly used in practice tries to achieve a kind of compromise between the two extremes. In order to do this the *mini-batch gradient descent* introduces a mini-batch of size m, with $1 < m < M$ where M is the size of the training set. Hence the gradient is calculated on the mini-batch what makes the procedure more stable than the SGD since the gradient is averaged over several training examples but with less computational and memory constraints compared to the batch gradient descent.

Nevertheless there remains the challenge to find a optimal learning rate as well as a suitable starting point to begin the descent procedure. For ANN we want to optimize the weights

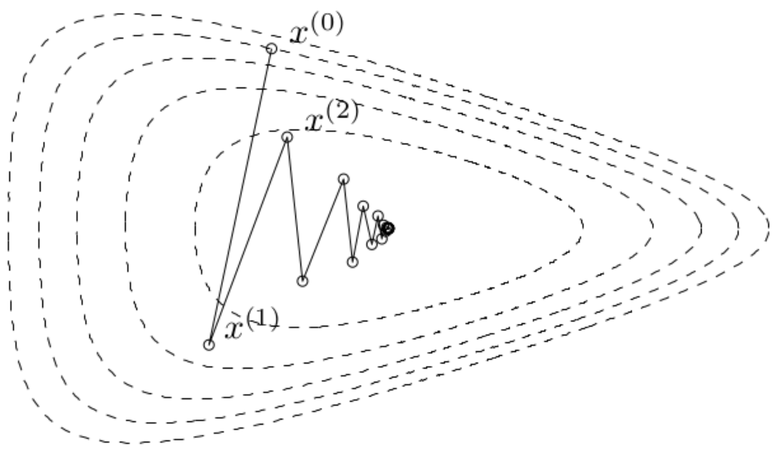

Figure 2.3: Zig-zagging behavior of the SGD (source: [BV04]).

w.r.t. the objective function and therefore the problem of finding a suitable starting point is translated to the problem of proper weight initialization. Although this problem is of utter importance for the convergence and final performance of the network it is not yet well understood because many characteristics of the general optimization of ANN are arbitrary and heuristic. Nevertheless several methods have been proposed that show encouraging results. One popular approach is the so called *xavier initialization* [GB10] that is described in 3.2.2.10.

The learning rate in contrast is often treated as a hyperparamter that is decreased in the training process (learning rate decay). Another approach is to let the optimization algorithm itself dynamically adapt the learning rate what brought forth several advanced training routines like *Adagrad* [DHS11], *Adadelta* [Zei12], *RMSprop* [TH12] and *Adam* [KB14].

Since it is the mostly deployed method of late and it will be used in the primary model of this thesis, we will take the time to shortly discuss Adam.

The basic idea of Adam what stands for ”adaptive moment estimation” is to use the first and second moment of the gradient to automatically schedule the learning rate. Therefore these moments are estimated by computing a exponentially decaying moving average over the preceding gradients, changing the update rule to:

$$x^{(t+1)} = x^{(t)} - \frac{\eta \; \hat{m}^{(t+1)}}{\sqrt{\hat{v}^{(t+1)}} + \epsilon}, \tag{2.27}$$

with the bias-corrected first moment estimate

$$\hat{m}^{(t+1)} = \frac{\beta_1 \cdot m^{(t)} + (1 - \beta_1) \cdot \nabla f(x)}{1 - \beta_1^{t+1}} \tag{2.28}$$

and the bias-corrected second moment estimate

$$\hat{v}^{(t+1)} = \frac{\beta_2 \cdot v^{(t)} + (1 - \beta_2) \cdot (\nabla f(x))^2}{1 - \beta_2^{t+1}} \tag{2.29}$$

where $t + 1$ when not in brackets means to the power of $t + 1$, what corresponds to the exponential decay guided by the decay parameter coefficients β_1 and β_2. This update procedure implicitly adds a *momentum method* [GBC16] to the gradient descent based learning process which further accelerates the convergence.

2.3.1.7 Architecture Design Decisions

The implementation and deployment of an ANN involves several decisions w.r.t both the used hyperparameters and the general underlying architecture.

Common hyperparameters that need to be selected are e.g. the number of training epochs, the dropout probability p, coefficients λ for the regularization and if no adaptive optimization algorithm is used the learning rate η as well. For this task usually an additional validation dataset or cross-validation is used. Then different search strategies can be employed to find suitable hyperparameters. These include e.g. the exhaustive *grid search* [CDM15] and *random search* [BB12] as well as advanced methods like *Bayesian search* [SLA12] or *evolutionary algorithms* [Ber+11]. Although such methods exist, recent complex and deep network structures include a lot of hyperparameters. This circumstance together with several possible choices for the network architecture (discussed in the following paragraph) leads to a combinatorial explosion that makes the training and optimization of ANN very expensive w.r.t. computation and memory capacity.

The architecture of an ANN depends on several decisions besides the choice of the general structure which includes feed forward neural networks, convolutional neural networks, RNN 2.3.2, GAN 2.3.3, Hopfield Nets [Hop82], Restricted Boltzmann Machines [SMH07], Self Organizing Maps [Koh82], Auto-Encoders [KW13] and many more. Moreover we have to specify both the depth of the network and the width of each consecutive or parallel layer. Finally respective activation and output functions have to be chosen.

Consequently the design and deployment of ANN remains a time consuming and highly non-trivial task that is suspect to ongoing research [GBC16].

2.3.2 Recurrent Neural Networks

Recurrent neural networks (RNN) [RHW86] are a specific type of ANN that is specialized to work on *sequential data.* More exactly this means data that is not independently identically distributed (i.i.d.) but consists of sequences of (x, y) pairs (where x is the data instance and y is the respective label) that show significant sequential correlation [Die02]. For the x values a sequence of length T is therefore given by $x^{(1)}, ..., x^{(T)}$.

In prediction tasks often the problem of *next item prediction* arises, where we want to predict the next element in the sequence and the respective label of $x^{(t)}$ is consequently given by $x^{(t+1)}$.

2.3.2.1 Networks for Sequence Modeling

The basic idea that makes it possible for ANN to process long sequences of data is to share its weights over several layers. If we take a sequence of length T and introduce a particular parameter for each time stamp t it wouldn't be possible to process sequences of a different length T' that wasn't encountered during training. The same is true for elements that can occur at different time stamps within the sequence. Our model should be able to recognize an element independent of it being at the beginning or the end of the sequence. That's why we share all the weights between the layers for each time step. Although this allows us to create a network that is independent of the sequence length, it forces us to alter the network structure and optimization procedure.

The essential part that enables us to share weights between layers and through time and that gave RNN its characteristic name is to define the network in a recurrent form [Gra12]. Therefore for this chapter each neuron represents a hidden state that we denote with h as well. This enables us to rewrite the network equation 2.12 as

$$h^{(t+1)} = f(h^{(t)}, x^{(t+1)}; \theta). \tag{2.30}$$

Generally this means that the hidden state $h^{(t+1)}$ depends on the preceding hidden state $h^{(t)}$ and the input data of time step $t+1$, given by $x^{(t+1)}$. Here f is a transition function that consequently is applied for each time step with the same parameters θ. The important idea is that most of the crucial information of the sequence up to time step t is accumulated in the state $h^{(t)}$ what is made possible by the recurrent formulation. Since the information of an arbitrary length sequence is represented as a finite fixed state, a certain loss of information has to be accepted. Nevertheless we assume that the most important aspects are captured by the network. Furthermore often an element at time step t doesn't necessarily depend on the whole sequence up to t but might primarily depend on the preceding last few elements in the sequence. The selection of essential aspects in the sequence data is guided by the continuous training process adapting the respective weights of the network.

2.3.2.2 Unfolding the Network

In order to use RNN for practical prediction tasks we usually add an output layer to the recurrent form. The way in which we add the output layer and the respective recurrent connections thereby essentially determines what type of RNN we obtain. Normally three general architectures are possible [GBC16]:

A) A network with a single output layer at the end that first processes the whole sequence by recurrently connected hidden states and then produces a final output.

B) A network that computes an output at each time step with recurrent connections between the hidden states.

C) A network that produces an output at each time step, however with the recurrent connection not between the hidden states but from the output of each time step to the subsequent hidden state.

Which type we want to use is mostly determined by the respective prediction problem. For the session-based recommendation task that is the focus of this thesis and that basically incorporates the problem of next item prediction we will use networks of type B). In figure 2.4 such a network is illustrated. The part on the left side represents the recurrent form where the black square indicates a delay of one time step. On the right side the unfolded computational graph of the RNN is depicted. That means that the individual state of each time step is shown for the whole sequence of length T. In the figure the output of each hidden state ($o^{(t)}$) is compared to the true label $y^{(t)}$ by a loss function denoted as $L^{(t)}$.

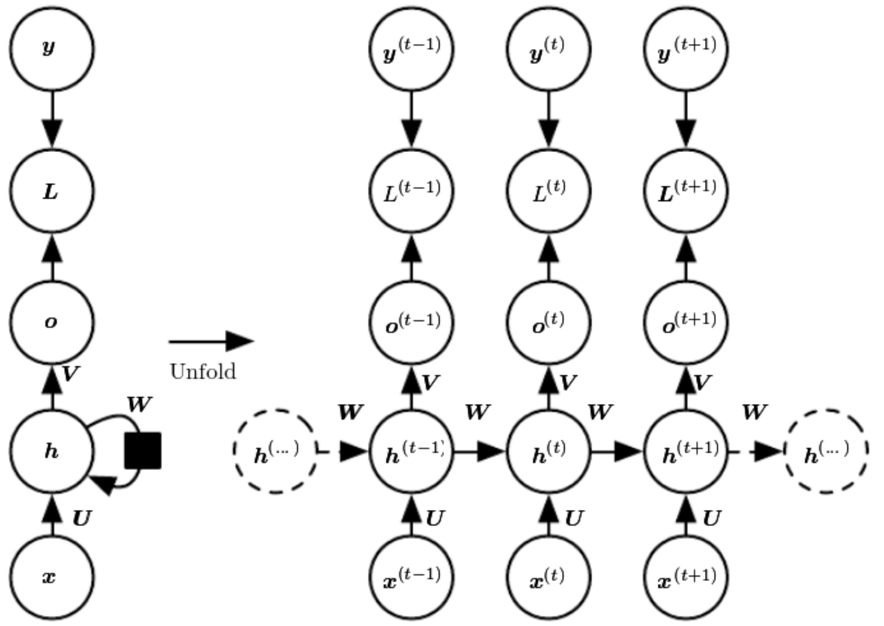

Figure 2.4: Unfolded representation of a RNN (type B) as computational graph (source: [GBC16], p.373).

More formally we can define the related forward pass by first introducing three weight matrices U, V and W that respectively govern the behavior of the input-to-hidden, hidden-to-output and hidden-to-hidden connections:

$$h^{(t+1)} = \phi_{hidden}(Wh^{(t)} + Ux^{(t+1)}), \tag{2.31}$$

$$\hat{y}^{(t+1)} = \phi_{out}(o^{(t+1)}) = \phi_{out}(Vh^{(t+1)}), \tag{2.32}$$

where ϕ_{hidden} and ϕ_{out} distinguish the hidden activation and the output function respectively. A common choice that promises a well behaved network for most standard tasks is the tanh for ϕ_{hidden} and the softmax function for ϕ_{out} since we're facing a classification setting. The total loss is normally calculated by summing over all loss functions.

The unfolded representation of the network has another advantage. It enables us to use the standard backpropagation algorithm 2.3.1.2 to compute the gradient. In the area of RNN this special procedure of unfolding the network graph and applying backpropagation is called *backpropagation through time* (BPTT). No other special training algorithms are required. Afterwards any general purpose optimization technique like the methods described in 2.3.1.6 can be used to update the parameters.

2.3.2.3 Bidirectional RNN

Until now the discussed RNN work on a sequence in chronological order from the first to the last element. However often an element within the sequence might depend on the whole sequence not just the preceding elements [Gra12]. For example in machine translation problems the actual translation of a word in a phrase will mostly depend on the sentence so far but as well on the subsequent parts. E.g. consider the following two phrases:

Peter hit the goal.
Peter hit the floor.

The meaning of the word *hit* in the two sentences is quite different, but this difference only becomes apparent with the last word. For this kind of problems where the causality is not necessarily dependent on the chronological order of the elements the class of *bidirectional RNN* (BRNN) has been developed [SP97].

The general idea of BRNN is to add another set of recurrent layers to the network that process each sequence in reverse order from the last to the first element. Furthermore both sets of recurrent layers are connected to the same output layer. In that way the network is able to capture complex dependencies of past and future context for a particular element in the sequence. Basically the forward pass stays the same, the only differences are that we now present the second hidden RNN layer with the input sequence in reverse order and the update of the output layer is delayed until both layers have processed the whole sequence. The same is true for the backward algorithm, that only needs to be adapted to calculate the total loss of all output layers and then to propagate it back through the hidden layers in opposite directions [Gra12].

2.3.2.4 Advanced Recurrent Units

Although RNN empower us to capture longtime dependencies within sequential data they have an inherent problem induced by their recurrent nature. This problem becomes obvious if we take a look on the gradients w.r.t. the deep computational graph of a RNN. What the RNN is basically doing is to apply the same operation with the same set of parameters repeatedly. For simplicity let the respective operation be given by a multiplication by the weight matrix W, then for a sequence of length t this is equal to multiplying by W^t. If we now pursue a Eigenvalue decomposition of the matrix W

$$W^t = (V diag(\lambda) V^{-1})^t = V diag(\lambda)^t V^{-1} \tag{2.33}$$

it becomes apparent that the Eigenvalues λ_i that have an absolute value smaller than 1 will vanish, while values larger than 1 will grow exponentially over time [PMB13]. The according problem therefore is called the *vanishing gradient* and the *exploding gradient* problem respectively.

To tackle this particular problem, advanced structures for the hidden units used in RNN have been developed:

- **Long Short Term Memory**
  The *Long Short Term Memory* (LSTM) [HS97] is a special type of hidden recurrent unit. It introduces self loops (self connections) with a fixed weight of 1.0 that support a constant flow of the gradient and therefore sometimes is also called a *constant error carousel* (CEC). Furthermore the flow of these self loops is controlled by another hidden unit, that "controls the gates" of the recurrent unit what led to naming them *gated units*. Basically the gated unit just stores the error gradients while the corresponding weights are responsible for the learning procedure. In order to control the error flow the gated LSTM cell has three gates:

  1. The *input gate* that determines if and how the state of the cell is altered by new information,
  2. the *output gate* which controls if and how the information stored in the hidden state is propagated,
  3. the *forget gate* that decides if the internal cell state should be totally reset by setting it to zero [GSC99].

  The whole structure of a single LSTM cell is depicted in figure 2.5. A more detailed discussion of the according error backpropagation and the training procedure can be found in [GSC99]; [GSS02]; [Gra12]

- **Gated Recurrent Unit**
  The *Gated Recurrent Unit* (GRU) [Cho+14a]; [Cho+14b] is a simplified version of the LSTM that needs less parameters to operate. This is achieved by combining the input and output gates of the LSTM in a single *update gate*. Figure 2.6 illustrates the general structure of the GRU.

  The activation of a GRU unit depends on the computation of the reset gate r given by [Cho+14a]:

$$r = \sigma(W_r x + U_r h^{(t)}), \tag{2.34}$$

  where σ is the sigmoid activation function described in 2.17 and the update gate z:

$$z = \sigma(W_z x + U_z h^{(t)}). \tag{2.35}$$

  The activation of the hidden unit h is then calculated by

$$h^{(t+1)} = z h^{(t)} + (1 - z)\tilde{h}^{(t+1)} \tag{2.36}$$

  with

$$\tilde{h}^{(t+1)} = \phi(Wx + U(r \odot h^{(t)})) \tag{2.37}$$

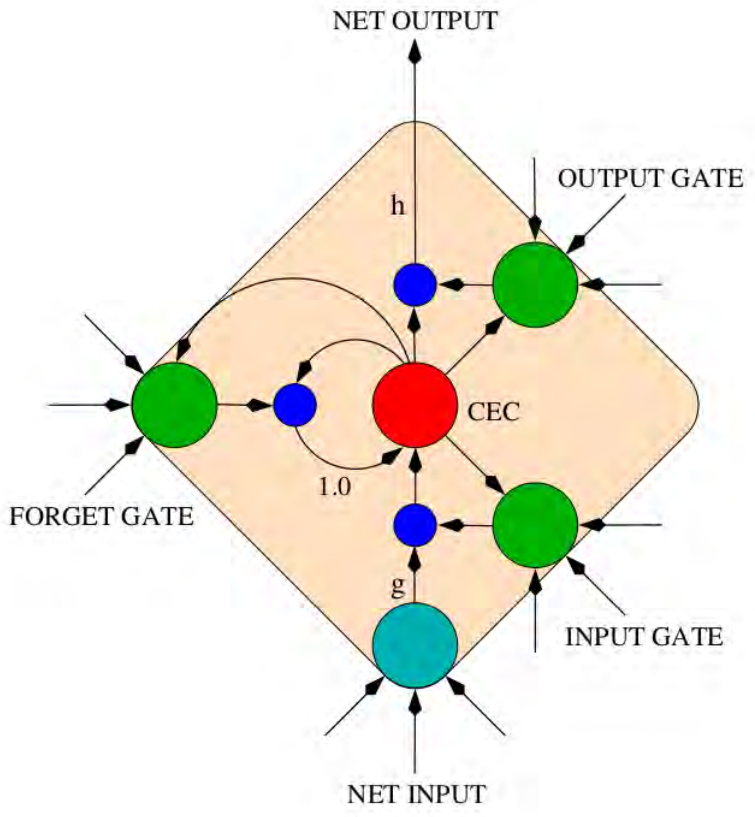

Figure 2.5: A basic LSTM cell with the respective control gates (source: [Gra12], p.34).

where ϕ as before is a nonlinear activation function and $\odot$ is the element-wise matrix product, also called the *Hadamard product* [Mil07]. This formulation allows the hidden state to ignore or even drop past information that is found to be irrelevant in the future completely when the reset gate is close or equal to zero. In contrary to the LSTM the update gate also controls how much information of the previous hidden state is transfered to the current state.

2.3.3 Generative Adversarial Nets

Generative adversarial nets (GAN) [Goo+14]; [Goo16a] are a special type of generative model that learns by simulating a game between two players. One player is the so called *generator*. It tries to generate samples $x = g(z; \theta_g)$ that resemble the samples of the true underlying distribution of the training data p_{data}. The other player is its adversary, the so called *discriminator* $d(x, \theta_d)$. It tries to distinguish between the generated fake samples of the generator and the real samples of the training data. So generally speaking the generator tries to fool the discriminator into thinking the generated samples were real while the discriminator tries to determine with high probability which samples are real and which are fake.

The respective game can simply be expressed as a *zero-sum game* [Bow09] where the payoff function of the discriminator is given by $v(\theta_g, \theta_d)$ and the respective payoff for the generator is simply $-v(\theta_g, \theta_d)$ [GBC16]. The training is guided by the according *mini-max*

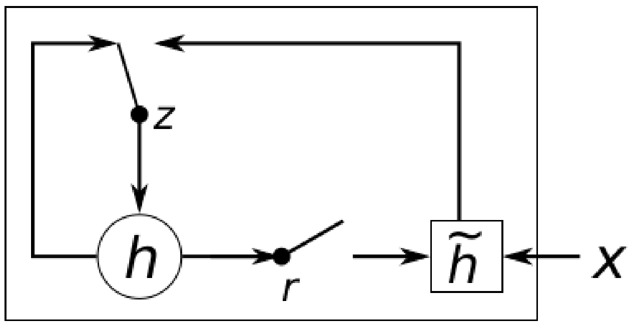

Figure 2.6: A basic GRU cell with the characteristic update and reset gates (source: [Cho+14a], p.3).

objective [OR94] that can be described as

$$g* = \arg\min_g \max_d v(g, d). \tag{2.38}$$

The natural solution to this game is a *Nash equilibrium* accomplished by the optimal set of parameters (θ_g^*, θ_d^*). The input to the generator is a noise sample z drawn from a simple prior noise distribution p_z. Then this input is transformed by the generator model resulting in a sample $x \sim p_g$. The ultimate goal of the learning process is to guide the generator by feedback from the discriminator to finally represent the real data distribution $p_g = p_{data}$ in the global optimum.

Usually g and d are arbitrary ANN, e.g. deep feed forward neural networks, RNN or CNN are possible candidates depending on the characteristic of the learning problem.

2.3.3.1 The Training Procedure

The training objective of a GAN is given by equation 2.38. For standard GAN the usual choice of the payoff function v is the cross-entropy:

$$v(\theta_g, \theta_d) = \mathbb{E}_{x \sim p_{data}} \log d(x) + \mathbb{E}_{x \sim p_g} \log(1 - d(x)). \tag{2.39}$$

The training consists of a sampling step in which two mini-batches of samples, one with the real values x and one constituting the generator input z from the noise prior are drawn. Then the generative transformation is computed and the respective loss is calculated. Next two simultaneous or rather alternating gradient steps are made to minimize the according payoff functions by updating the parameter sets θ_g and θ_d. Any arbitrary SGD based optimization routine can be employed, usually Adam shows good results [Goo16a].

At convergence $p_g = p_{data}$ and the samples of the generator are indistinguishable from the real samples. Consequently the discriminator assigns a probability of $\frac{1}{2}$ to all real and fake samples.

Although these theoretical properties are promising, training GAN in practice is quite difficult. The difficulty is mostly caused by the non convexity of $\max_d v(g, d)$ that can prevent the convergence of the GAN what usually leads to *underfitting* the data [Goo14]. Moreover there is no theoretical guarantee that the game will reach its equilibrium when using simultaneous SGD methods. Thus the stabilization of GAN training remains a open research issue. Careful selection of hyperparameters shows reassuring results w.r.t. to the training and stability of GAN, however the hyperparameter search is often computationally restrictive for complex networks.

2.3.3.2 Energy-based GAN

Energy-based GAN (EBGAN) [ZML16] is a alternative formulation of the GAN architecture. The advantage that is achieved by EBGAN is that much more types of loss functions can be employed compared to the standard GAN that normally only allows to use the binary cross-entropy. To be able to do this the discriminator is interpreted as an energy function [LeC+06]. This function maps each point in the input space to a scalar. So basically the discriminator is now used to allocate low energy to regions near the data manifold and high energy to other regions. In this way the energy mass is shaped in the training ultimately forming the surface of the underlying distribution. We can also think of the discriminator as being a parameterized and therefore trainable cost function for the generator. In contrast the generator learns to produce samples that belong to regions which get assigned low energy by the discriminator.

As before the discriminator is served with mini-batches of real and fake samples. Then the discriminator estimates the according energy value for each sample by means of a loss function $\mathcal{L}$. Several loss functions are possible, Zhao et al. [ZML16] use a *general margin loss* that is given by:

$$\mathcal{L}_d(x, z) = d(x) + \max(0, m - d(g(z))), \tag{2.40}$$

where m is the margin that is treated as a hyperparameter and consequently the loss for the generator is:

$$\mathcal{L}_g(z) = d(g(z)). \tag{2.41}$$

The duty of the margin is to separate the *most offending incorrect examples* [LeC+06] from the rest of the data. The most offending incorrect examples are samples that were incorrectly classified and lie very far from the decision boundary, meaning these are the examples with the lowest energy among all incorrect examples. All other examples contribute a loss of 0. In this way the energy surface can be shaped effectively.

2.3.3.3 Sequential GAN

There is one big issue with the standard GAN: It only works for continuous data. Although e.g. compared to *Variational Autoencoders* [GBC16] there is no problem with discrete hidden units in the network, it doesn't work out of the box for discrete output. The reason for this is twofold.

First, in case the generator produces discrete output it is very difficult to make a particular gradient update for the generator [Goo16b]. The general problem is the size of the gradient steps. Basically our generative model is updated by computing the error gradient by the discriminator w.r.t. the produced fake samples, this means we get information on how to slightly change the sample to better fool the discriminative model the next time. For continuous data, e.g. an image, this means for example that we change the value of a pixel by adding 0.0001 to it. If we do this for a huge amount of samples we continuously improve the quality of the fake image to closer match the real images. However for a discrete sample, e.g. a word in a NLP model, we cannot just add a small value to it. Even if all the words are given as an embedding, there just might be no element corresponding to the new numeric value and consequently the update procedure doesn't lead to verifiable improvement.

Secondly, when speaking about sequential data we need to decide how the score of a partially generated sequence should be weighted w.r.t. the finished sequence while training. This task is necessary to calculate the corresponding loss but is non-trivial.

To alleviate these issues and render GAN applicable to sequential data [Yu+17] introduce *Sequential GAN* (SeqGAN). Their main contribution is to use reinforcement learning (RL) [SB98] with a stochastic policy gradient to approximate the respective gradients.

Therefore the generator is treated as a RL agent working on a policy whose respective state s_t is given by the partially generated sequence $\hat{y}_1, ..., \hat{y}_t$ at time step t and an action $a \in Y$, where Y is the available vocabulary, corresponding to the next element in the sequence $\hat{y}_{t+1}$. Note that we changed the notation from superscript (t) to subscript t to be conform with the standard RL notation. Accordingly the generator represents a stochastic policy model:

$$g(\hat{y}_{t+1}|\hat{y}_1, ..., \hat{y}_t; \theta_g). \tag{2.42}$$

However fortunately the transition of a state s_t to the next state s_{t+1} is deterministic, since choosing an action a leads to a transition

$$\delta^a_{s,s'} = p(s_{t+1} = s'|s_t = s, a_t = a) = 1 \tag{2.43}$$

whereas

$$\delta^a_{s,s''} = p(s_{t+1} = s''|s_t = s, a_t = a) = 0, \quad \forall s'' \neq s'. \tag{2.44}$$

The update procedure for the generative model involves the respective policy gradient in conjunction with a *Monte Carlo* (MC) search [Bro+12]. Hence the objective of the generator is to produce a sequence beginning in an initial state s_0 that maximizes the expected final reward:

$$\mathbb{E}[R_T|s_0; \theta_g] = \sum_{\hat{y}_1 \in Y} g(\hat{y}_1|s_0; \theta_g) \cdot Q(s_0, \hat{y}_1) \tag{2.45}$$

where R_T depicts the reward that is achieved for a complete sequence and $Q(s_0, \hat{y}_1)$ is the corresponding *action-value function* [Yu+17]. This function represents the total expected reward by following the policy g starting with an initial state s_0 and then consecutively taking action a. The ultimate goal of the agent given by the generator is to produce a sequence evolving from the initial state that is considered real by the adversary. Thus we are left with finding a method to estimate the action-value function for this particular problem. A standard choice for the general task is the REINFORCE algorithm proposed by [Wil92]. Since the discriminator estimates the probability of a sequence being real, it can be directly employed as an approximate estimate of the action-value function:

$$Q(s = \hat{y}_1, ..., \hat{y}_{T-1},\ a = \hat{y}_T) = d(\hat{y}_1, ..., \hat{y}_T; \theta_d) \tag{2.46}$$

The remaining issue is that depending on the task the discriminator is only able to provide a reward value w.r.t. finished sequences. Moreover since we are interested in the longterm reward for a finished sequence that is able to fool the discriminator, we need to focus on this goal on every time step by considering the resulting future outcome. It might be advantageous to sacrifice some intermediate interests for a higher final reward. To incorporate this behavior in the described approach a MC search in conjunction with a *roll-out policy* [Gel+12] is performed. Here the roll-out policy samples the last $T-t$ missing elements to complete a sequence. In order to obtain a sufficiently accurate estimate this is done for N times resulting in a N-Time MC search [Yu+17]:

$$Q(s = \hat{y}_1, ..., \hat{y}_{T-1},\ a = \hat{y}_T) = \begin{cases} \frac{1}{N} \sum_N d(\hat{y}_1^{(n)}, ..., \hat{y}_T^{(n)}; \theta_d) & for \quad t < T \\ d(\hat{y}_1, ..., \hat{y}_T; \theta_d) & for \quad t = T \end{cases} \tag{2.47}$$

where $\hat{y}_1^{(n)}, ..., \hat{y}_T^{(n)}$ are the completed samples of the roll-out policy in the n-th MC search. The general procedure is illustrated in figure 2.7.

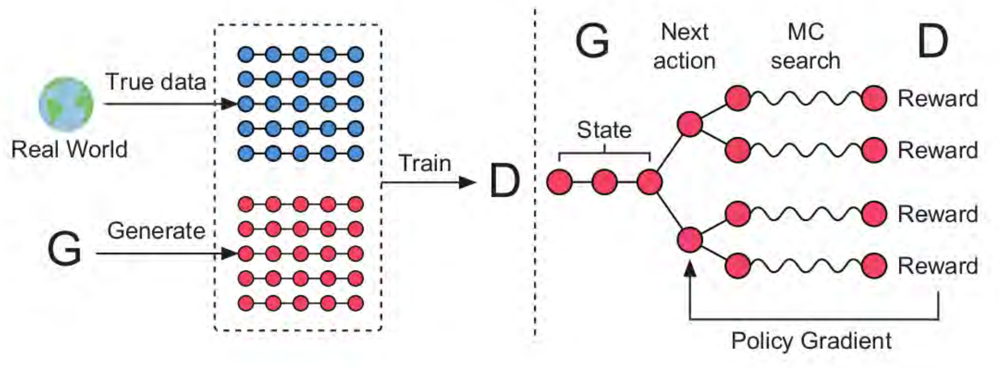

Figure 2.7: The SeqGAN training procedure (source: [Yu+17], p.3).

Here the generator and discriminator are depicted respectively by **G** and **D**. Note that the general training procedure for the discriminator basically stays the same. As shown on the left of figure 2.7 it is fed with real data and fake samples and alternately trained according to

$$\min_{\theta_d} -\mathbb{E}_{x\sim p_{data}} \log d(x) + \mathbb{E}_{x\sim p_g} \log(1 - d(x)), \tag{2.48}$$

which combines equations 2.38 and 2.39 for the discriminator.

However an important difference to standard GAN training is that in most cases it is recommended to pre-train the discriminator and optionally the generator model before starting the adversarial training procedure to support faster convergence.

3. Design and Development

In this chapter I first take some time to give an overview of the problem that was already shortly discussed in 1.2. Then I present the details of the model I propose to solve the identified problem. Therefore we will have a look on the respective theoretical foundations, the building blocks of the model and the challenges w.r.t. its implementation.

3.1 Problem Overview

The focus problem of this thesis is the general recommendation problem of 2.1 but with several restrictions and additional challenges described in 2.1.5. So we still want to leverage data on users and their preferences to predict their possible future likes and interests, however the data is restricted to the clickstream in a session of a particular user. We therefore also call this a *session-based* recommendation setting. That means the data consists of a sequence of items $x_1, ..., x_t$ that a user visited or viewed consecutively up until the time step t. Since the user is not required to be logged in and is not holistically tracked, a user cannot be identified and consequently no profile or historical data about the user is available. This very restrictive data environment severely reduces the performance of most of the traditional CF approaches [Hid+15]. Instead one can fall back on content-based RS using item-to-item similarity. Another possibility is given by probabilistic models based on item co-occurrence like *association rules* (AR) [AIS93] or transition probabilities, e.g. *Hidden Markov Models* (HMM) [Cao+09]; [SSM12]. However these models often produce inferior results compared to the standard recommendation task.

3.2 Proposed Model

Another approach to tackle the aforementioned problem is to use RNN to capture the underlying sequential nature of the data directly. This was first proposed by [Hid+15]. Their model that is described in 4.3.3 achieved surprising state-of-the-art results in the session-based recommendation task. Therefore an adapted version will serve as one of several reference models to evaluate the model implementation in this thesis.

The general architecture for the proposed model is based on the ideas presented in [Yoo+17]. To the best of my knowledge it is the first work to propose a GAN architecture for recommendation that combines the ideas of EBGAN (2.3.3.2) and SeqGAN (2.3.3.3) in a model called *EB-SeqGAN*. However the authors take a rather theoretical perspective and for example describe the connection of EBGAN to *imitation learning* instead of forcing a practical implementation with problem specific challenges and a final comparative evaluation. Therefore this thesis is concerned with adapting and refining the presented ideas to create a fully functional prototype model and evaluate it against several reference models to demonstrate its capabilities. Following the RNN-based recommendation model by [Hid+15] called **GRU4Rec** I name the model proposed in this thesis **GAN4Rec**.

3.2.1 Training Objective

Since the EB-SeqGAN architecture is governed by the EBGAN framework to obtain the objective functions for the generative and discriminative model of GAN4Rec we start with the EBGAN objective [ZML16]:

$$\min_{d} \quad \mathbb{E}_{x \sim p_{data}} d(x) + \mathbb{E}_{x \sim p_g} \max(0, m - d(x)) \tag{3.1}$$

and

$$\min_{p_g} \quad \mathbb{E}_{x \sim p_g} d(x) + \psi(p_g). \tag{3.2}$$

where ψ is an optional regularization term. However we have to take care of the characteristics of the SeqGAN and therefore need to employ the policy gradient technique to train the generator. Since we want to minimize the loss w.r.t. the parameters of the generator we alter the formulation of the expected reward 2.45 to give the *expected total loss* (ETL)

$$\mathbb{E}[\hat{\mathcal{L}}|x;\theta_g] = g(\hat{Y}|x;\theta_g) \cdot Q_{\mathcal{L}}(x,\hat{Y}). \tag{3.3}$$

In analogy to 2.47 $Q_{\mathcal{L}}$ can be estimated with a N-time MC search involving a roll-out policy and $d(\hat{Y})$:

$$Q_{\mathcal{L}}(x,\hat{Y}) = \begin{cases} \frac{1}{N} \sum_N d(\hat{Y}^{(n)}_{1:T-1};\theta_d) & for \quad t < T \\ d(\hat{Y}_T;\theta_d) & for \quad t = T \end{cases} \tag{3.4}$$

This leads to the final formulation of the generator objective

$$\min_{p_g} \quad g(\hat{Y}|x;\theta_g) \cdot Q_{\mathcal{L}}(x,\hat{Y}) + \psi(p_g). \tag{3.5}$$

3.2.2 Design Decisions

In the following section the design decisions for the model are described and justified.

3.2.2.1 Adversarial Training

As mentioned, [Hid+15] proposed the first RNN model for session-based recommendation. To be able to train this model effectively they introduced a new ranking loss function that was further improved by their subsequent work [HK17].

In case of the SeqGAN architecture there is the advantage that the payoff or respective loss function is given by the discriminator model and consequently is parameterized and adaptive since it can be dynamically updated and refined in training. Thus the task of designing a suited loss function for the generative model is obsolete. Although we still need to find suitable loss functions for the generator and discriminator, these are generally much simpler and often can be derived from the general network and problem structure.

An advantage of the EBGAN framework is that its implicit objective is to learn and model the underlying unknown distribution of the data. In this scenario the generator is punished when producing samples outside the underlying data manifold, what can be interpreted as training the generator with a set of possible labels (part of the whole data manifold) in contrast to single outputs in standard supervised learning. This approach enables the model to incorporate an enormous generalization capability when it is executed well enough and is trained to sufficient convergence. Furthermore e.g. compared to HMM based models adversarial networks are able to capture very sharp or even degenerate distributions that might be characteristic for some data [Goo+14].

3.2.2.2 EBGAN and Imitation Learning

According to [Yoo+17] and [Fin+16] there is a connection between the learning procedure of energy-based GAN and imitation learning [Zie+08], sometimes also called *apprenticeship learning*. Both terms describe approaches that involve *inverse reinforcement learning* (IRL) that basically is concerned with inferring the utility function induced by a certain behavior. The general idea is to find the policy that most closely mimics the behavior of an expert that is assumed to be nearly optimal. This idea can be projected on our session-based recommendation problem by interpreting the finished sequences of the training data as the expert behavior. Consequently we now want to estimate the policy which most closely mimics this expert behavior, in other words the policy that best recovers the underlying unknown distribution of the data. Hence our method is a reasonable approach to design a RS. A detailed discussion and the respective proofs can be found in [Yoo+17].

3.2.2.3 Application of RNN

In contrast to the SeqGAN model proposed by [Yu+17] that employs a RNN as generator and a CNN for the discriminator, GAN4Rec introduces a bidirectional RNN for the discriminator. The reasons for this are twofold:

1. The first reason is the fact that RNN were especially designed and developed to work on sequential data. They show state-of-the-art performance for several sequence-based problems [Mik+10]; [SVL14]; [Hid+15]. So there is no reason why we shouldn't leverage these capabilities for the discriminator as well, since it has to deal with the same sequential data as the generator. Furthermore, because the discriminator is going to process only complete sequences, a bidirectional form is a reasonable choice to capture the context for each element over the whole sequence.

2. Secondly the examples in [Yu+17] only demonstrate the performance for generated sequences of fixed length. However this is a characteristic that is not common for most real data. The authors argue that max-over-time-pooling can be used to extent CNN to work for variable-length sequences. However [RMC15] show that max-pooling layers can lead to slow convergence of GAN and therefore should be avoided or rather replaced with strided convolutions. In contrast the inherent structure of RNN is designed to naturally work for sequences of varying length.

3.2.2.4 Model Regularization

Concerning the regularization of the model, [Yoo+17] propose to replace the heuristic *repelling regularizer* in the generator objective function of the standard EBGAN [ZML16] with the *negative entropy* (cf. 2.7):

$$-H(p_g) = \sum_{i}^{N_c} \hat{y}_i \log \hat{y}_i = -\sum_{i}^{N_c} \hat{y}_i \log \frac{1}{\hat{y}_i}, \tag{3.6}$$

where N_c is the vocabulary size given by the number of classes or categories and $\hat{y}_i$ is the estimated probability of class i.

The authors motivate this choice with the regularization of the generator distribution p_g to be less sharp when the entropy becomes smaller and therefore to prevent *mode collapse*. Mode collapse is a common issue when training GAN on real data. It describes the behavior of the generator to concentrate most of the probability mass in some limited regions [Goo16a]. This happens because the generator finds a particular setting that is accepted and labeled as true by the discriminator and therefore starts to exclusively use this successful setting. The problem with this behavior is that although only a small subset of the possible sampling space is covered the training does not lead to further improvement and the distribution converges to a quite suboptimal equilibrium.

However I argue that just adding the negative entropy to the objective function of the generator model might deteriorate the training in practical implementations. If we take a look on the definition of the entropy, it normally is defined for a multinomial distribution with a class probability vector $\vec{p}$ [Sha01]:

$$H(\vec{p}) = -\sum_{i}^{n} p_i \log p_i = \sum_{i}^{n} p_i \log \frac{1}{p_i}. \tag{3.7}$$

The entropy term gets maximal for a uniform distribution p_u

$$H(\vec{p_u}) = n \cdot \frac{1}{n} \log n = \log n, \tag{3.8}$$

and small when most of the probability mass is concentrated in one point ν

$$H(\vec{p_\nu}) = 1 \log 1 = 0. \tag{3.9}$$

The entropy is therefore non-negative and its lower bound is zero [Car14]. Adding the negative entropy $-H(p)$ as regularizer term to our objective function 3.5 yields

$$\min_{p_g} \quad g(\hat{Y}|x;\theta_g) \cdot Q_{\mathcal{L}}(x,\hat{Y}) - H(p_g). \tag{3.10}$$

Accordingly our loss is reduced when p_g is close to uniform and in contrast nearly stays the same when incorporating sharp modes. From a theoretical point of view this behavior might be preferable. However from a implementation perspective there are some important properties we need to take care of. In the above formulation the lower bound of $-H(p)$

is given by $-\log n$ [Car14] so depending on the vocabulary size it is possible for the loss to become negative when the reward term is overpowered by the regularizer. Furthermore we also need to take care of possible scale differences between $d(x)$ and $-H$ that might have a high impact on training and convergence speed.

To alleviate these issues I propose a new regularization term called *normalized inverse entropy* (NIE) :

$$\bar{H}^{-1}(\vec{p}) = \frac{\log n}{\sum_i^n p_i \log \dfrac{1}{p_i}} \tag{3.11}$$

Since the standard entropy H is not invertible this rather corresponds to its reciprocal value function that in the implementation is stabilized by adding a small ϵ, e.g. $\epsilon = 10^{-21}$ to avoid precision overflow by $\log 0$.

This new regularization term is non-negative and becomes small when the according distribution is near uniform and large when it is sharply peaked. Moreover it is normalized by the log number of classes and therefore its magnitude doesn't heavily depend on the vocabulary size. The negative entropy and normalized inverse entropy are plotted against each other in figure 3.1 for several vocabulary sizes.

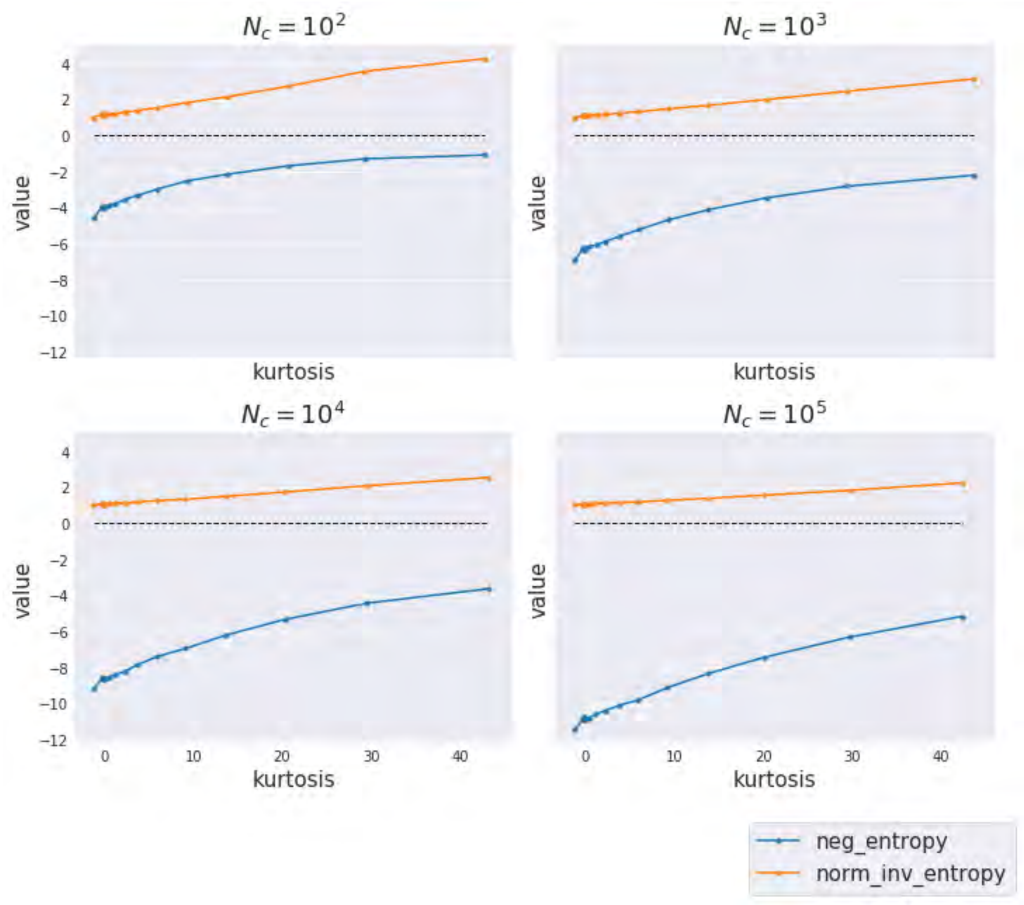

Figure 3.1: Plots of the negative entropy and normalized inverse entropy for several vocabulary sizes N_c.

The shape property of the unimodal probability distributions from which was sampled was measured by the *kurtosis* which caracterizes the 4th moment of a distribution and is defined by [Wes14]:

$$kurtosis = \frac{\mathbb{E}[(X-\mu)^4]}{\sigma^4 - 3} \tag{3.12}$$

As we can see in figure 3.1 $-H(p)$ is always negative and the magnitude of the curve changes with N_c. Furthermore we see a high difference between the first and the second measurement point that marks the transition from a uniform distribution with maximal entropy to an approximately normal distribution in the area with a kurtosis close to 0. At the same time the normalized inverse entropy increases nearly linearly without large changes of magnitude.

Although the investigation and derivation of detailed mathematical properties for the normalized inverse entropy would be an important and interesting topic, it is beyond the scope of this thesis.

Nevertheless the new regularizer allows us to add it to 3.5 without risking a negative loss. Consequently our objective for the generator becomes

$$\min_{p_g} \quad g(\hat{Y}|x;\theta_g) \cdot Q_{\mathcal{L}}(x,\hat{Y}) + \lambda \bar{H}^{-1}(p_g), \tag{3.13}$$

where $\lambda \geq 0$ is the regularization coefficient that is treated as a hyperparameter.

Additional regularization is achieved by respectively applying the dropout technique explained in 2.3.1.5 to the generative and discriminative model.

3.2.2.5 Loss Function

Following the remarks in 2.3.1.7 the choice of a suited loss function is a crucial part of the model design. As was already explained in 3.2.1 and 3.2.2.1 the loss of the generator is directly given by the discriminative model applied to the generator output. The comprehensive loss estimation is achieved by the respective RL policy framework with MC search.

For the discriminator several reasonable loss functions are possible. The authors of [ZML16] employ a simple general margin loss 2.40. However I experiment as well with the application of the *square-square loss* [LeC+06] that for the respective model is given by

$$\mathcal{L}_d(x,z) = d(x)^2 + (\max(0, m - d(g(x_0))))^2. \tag{3.14}$$

where d and g are the discriminator and generator models, x_0 is the start token for the generator to sample from and $m > 0$ is the according margin.

Moreover I investigate the effect of decaying the margin in training by multiplying it by a decay factor $\gamma \in (0,1)$.

3.2.2.6 Embeddings

The general task of embedding methods is to create a mapping $\mathbb{C}^m \rightarrow \mathbb{R}^n$ from a discrete space to a dense continuous vector space where usually $m >> n$. Embedding techniques originate in computational linguistics and NLP approaches to reduce the dimensionality of words in conjunction with their contexts. The goal is to create a low dimensional representation that preserves high-level dependencies in its structure. This means an embedding tries to compress the data while creating an expressive representation. A well known example is the *word2vec* technique developed by [Mik+13]. This method maps words into a metrical space where dependencies of words can be expressed by their distance and relative position. Accordingly words that often occur together share a region within the n-dimensional space. Mikolov et al. showed that the embedding even enables practical inference by using vector arithmetics in the subspace. In the famous example they use a vector operation consisting of the numerical vectors for "king", "man" and "woman" to arrive at a vector that is most similar to the vector of the word "queen". This shows that embeddings are able to conserve and even enhance existing dependencies in the training data. Moreover ANN have shown especially good results in producing high quality embeddings [MH09]. Consequently I use embeddings to create meaningful representations from the discrete item sequences that can be fed directly into the respective models. Therefore both the generative and the discriminative model receive an embedding layer. This layer or rather its weights are trained directly in conjunction with the rest of the model in the pre-training stage (cf. 3.2.4). Then the embedding is fixed for the adversarial training.

3.2.2.7 Gradient Stabilization

Since the proposed model is meant to work on item sequences given by click streams, the length of the respective sequences can be arbitrarily large. Although sequences that highly deviate from the average should be treated as outliers, the sequence length is still large compared to classical NLP models working on sentences. For this reason the inherent problem of exploding and vanishing gradients has to be addressed. As discussed in 2.3.2.4 the deployment of advanced recurrent units can prevent the vanishing gradient behavior for the most part. Therefore I use GRU cells for both the RNN in the generative and in the discriminative model. I prefer the GRU over the LSTM architecture since it achieves comparable results [Cho+14a]; [Cho+14b] while at the same time providing a less complex structure with fewer parameters.

However compared to the usual application of RNN I am going to use them in a RL setting that involves the calculation of policy gradients for complete sequences (that consequently have maximum length) for n-times. This special application apparently leads to exploding gradients that are not affected by the usage of GRU, some of which were encountered in training. Therefore I suggest the additional employment of advanced techniques for gradient stabilization. These techniques include the *gradient norm* and *gradient clipping* [PMB13] methods. The gradient norm takes the norm of the gradients, replacing the standard gradient in the update rule according to

$$\nabla f = \|\nabla_w \mathcal{L}\|_\Omega. \tag{3.15}$$

where $\|\cdot\|_\Omega$ is the respective norm. I basically use the L2 norm however the infinity norm is also possible.

In the next step the gradient clipping technique constrains the maximum value of the gradient norm to a predefined threshold value τ_{max} that cannot be exceeded, consequently globally preventing the explosion of gradients:

$$\nabla f = min(\|\nabla_w \mathcal{L}\|_\Omega \ , \ \tau_{max}). \tag{3.16}$$

τ_{max} is treated as another hyperparameter to be found empirically. Furthermore for convex settings the gradient clipping technique can be interpreted as an additional regularization method that restricts the gradient to the unit ball of the respective norm [Mer+16].

3.2.2.8 Ranking Approach

In most recommendation scenarios we are not just occupied with recommending a single best item to the customer but rather with producing a whole list containing the possibly most interesting item suggestions [AT05].

To this end we need to design a suited ranking approach for GAN4Rec to be able to create lists of items instead of single predictions. Running the model several times on the same sequence and hoping for different predictions of varying confidence is no reasonable approach. Fortunately a ranking method can be easily and directly incorporated in the model structure. The generator output basically is a probability vector over the respective N_c classes. Therefore the according log probabilities can be interpreted as preference scores for each particular item. For this reason we can directly employ these scores for a ranking scheme that brings the items in a descending order depending on their scores and then simply selects the top-N items for recommendation.

3.2.2.9 Optimization

As optimization algorithm for the GAN4Rec model I use the Adam method discussed in 2.3.1.6. The task of this optimizer is to update the weights by employing the loss gradient and a suited learning rate η that is adaptively changed each iteration. However the algorithm still needs a start value η_0 for the dynamic adaption. Common values are 10^{-2} and 10^{-3} while for deep convolutional GAN a value of $2 \cdot 10^{-4}$ was reported to produce better results [RMC15]. Although the Adam optimizer achieves reasonable performance all by itself, my experiments show that an additional *simulated annealing* procedure for η_0 can further improve the results. In this procedure the learning rate is reduced in each outer iteration of the adversarial loop, by multiplying it with a scaling factor $\xi \in (0, 1)$. The improvement is possibly due to the characteristic alternating training of the GAN framework in which there are two independent optimizer instances used for the generator and the discriminator. Since in each iteration the generator and the discriminator are individually trained and updated while the other part of the model is ”frozen”, the situation at the end of an individual training iteration and when it is continued is subject to change. Therefore it seems to be beneficial to reinitialize the optimizer for each iteration but choosing a slightly smaller η_0 each time to compensate for the progress in training.

3.2.2.10 Initialization

[GB10] studied the behavior of backpropagated gradients in deep network architectures and its dependence on different activation functions, neuron saturation and initialization. Their results suggest that pure random initialization and even heuristic initialization procedures that sample the weights from an adapted uniform distribution u

$$W^{(l)} \sim u(-\frac{1}{\sqrt{N_{l-1}}}, \frac{1}{\sqrt{N_{l-1}}}), \tag{3.17}$$

where N_{l-1} is the size of the preceding layer, can lead to increasingly small gradients the farther they are backpropagated through the network. This behavior once again results in slow convergence of the network during training. To alleviate the corresponding problem the authors propose the *normalized initialization*, to which in the ANN community often is referred to as *Xavier initialization* because of the surname of the first author Xavier Glorot. The proposed initialization routine is given by

$$W^{(l)} \sim u(-\frac{\sqrt{6}}{\sqrt{N_{l-1}+N_l}}, \frac{\sqrt{6}}{\sqrt{N_{l-1}+N_l}}). \tag{3.18}$$

The notable idea of this approach is to force the variance of the gradients in the preceding and subsequent layer to be close and in this way to ensure that the gradients do not vanish (or explode) over time. However this effect becomes weaker the further the training progresses from its initial state. A detailed discussion can be found in [GB10].

Moreover the same method can be applied to a Gaussian distribution. Then the sampling distribution for the weights becomes

$$W^{(l)} \sim \mathcal{N}(0, std_{xavier}), \tag{3.19}$$

where

$$std_{xavier} = \sqrt{\frac{2}{N_{l-1}+N_l}}. \tag{3.20}$$

3.2.3 Model Components

Since every GAN model is made up of at least one generative and one discriminative model part, the according components for GAN4Rec are explained below in detail.

3.2.3.1 Generator

The generator for the GAN4Rec model consists of an embedding layer that is directly followed by the RNN represented by a GRU layer. This in turn leads to the final fully connected linear output layer which is employing a softmax output function to map into the discrete N_c-dimensional item space. An illustration of the generator model is shown in figure 3.2. The properties of the particular layers are presented in table 3.1. I experimented with different dropout rates for the hidden and the embedding layers, however dropout in the embedding always led to a decrease in performance.

Layer	**Generator**	**Discriminator**
Embedding dim	32	64
Dropout prob embedding	0.0	0.0
Hidden dim	512	128
Dropout prob hidden	0.4	0.2
Num GRU layers	1	2
Num Linear layers	1	2

Table 3.1: Dimensions and dropout probabilities of generator and discriminator layers.

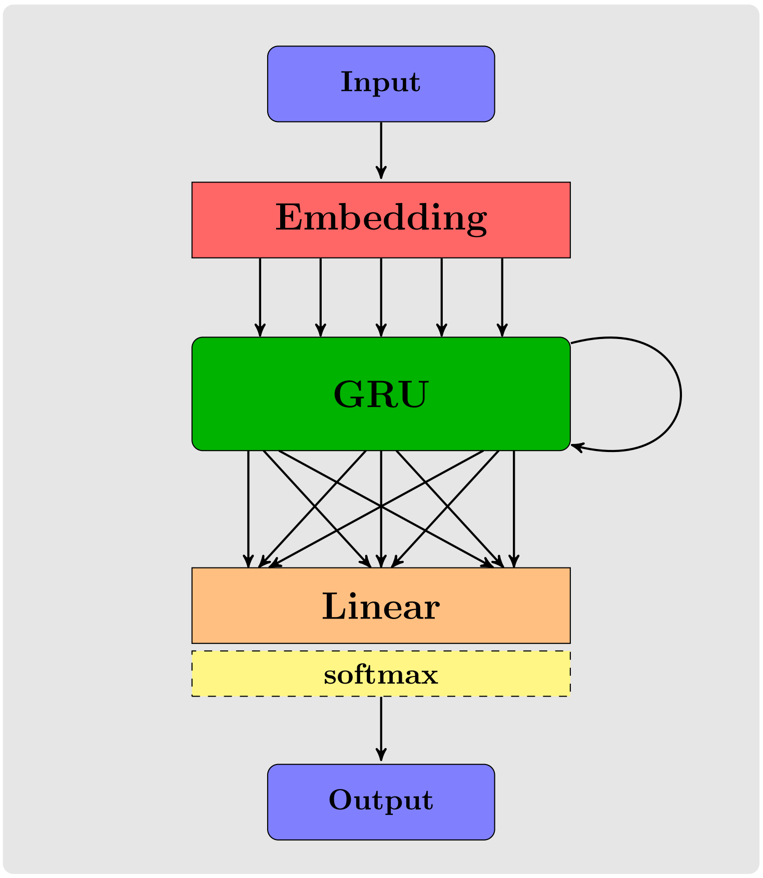

Figure 3.2: Architecture of the generative model.

3.2.3.2 Discriminator

Compared to the generator the discriminator has a more complex structure. It consists of a primary embedding layer that is followed by two bidirectional GRU layers. The latter feeds again into two consecutive fully connected linear layers, where the first one employs a leaky ReLU activation function and the second uses a sigmoid to represent the binary decision problem of determining which samples are real and which are fake. For the hidden activation I choose the Leaky ReLU because it has been shown to achieve good performance in GAN models and helps to avoid sparse gradients [RMC15]. The slope parameter λ is set to a value of 0.2. The layer properties are listed in 3.1 and an illustration of the general architecture of the discriminative model is depicted in figure 3.3.

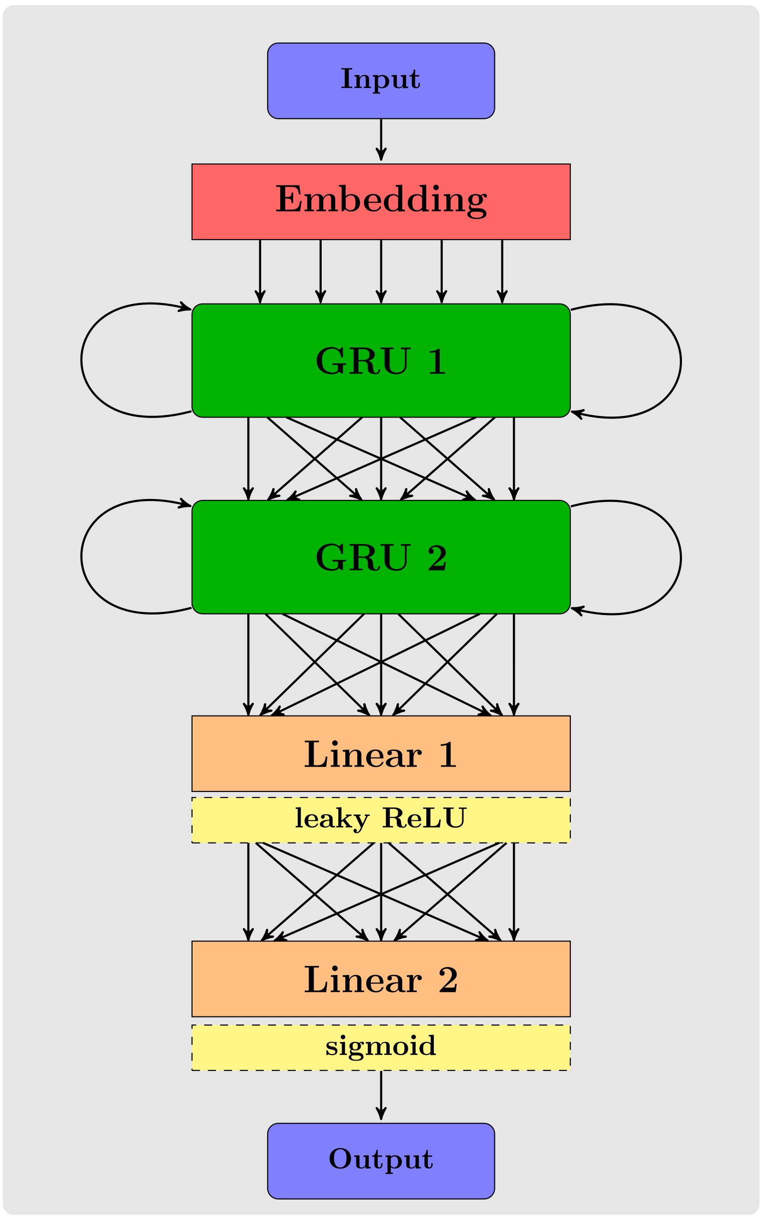

Figure 3.3: Architecture of the discriminative model.

3.2.4 Model Training

Besides the architectural design the training procedure is one of the crucial parts of developing a practical ML model. Yu et al. show in [Yu+17] that it is beneficial for the adversarial training to have a prior step in which the generator and discriminator model are prepared. It seems to be especially helpful to have a pre-trained discriminator that is able to provide an informative training signal to the generator at the start of the adversarial training. This approach supports faster convergence. Another advantage is that in this way the embeddings for the sequences can be pre-trained in conjunction with the model. Therefore the training of GAN4Rec can generally be divided into three consecutive steps, the *generator pre-training*, the *discriminator pre-training* and the *adversarial training* stage. The details of these stages are explained below.

3.2.4.1 Generator Pre-Training

Prior to the pre-training of the discriminator we need to initialize the parameters of the generative model. However the untrained generator cannot provide good contrastive samples. This is due to the fact that in this case the generator basically produces random samples since it has not yet learned to mimic the true DGP. For this reason it won't be hard for the discriminator to distinguish between the produced fake samples and the real data. Thus the need arises to pre-train the generator for some iterations before starting the discriminator pre-training.

For the pre-training of the generator I basically follow the strategy of [Yu+17]. However I initialize the model by Xavier normal initialization (Eq. 3.19). Then it is trained via maximum likelihood estimation. Therefore the generator receives batches of sequences and then for each element is trained to predict the subsequent element. As loss function a negative log likelihood loss is employed. Normally the model is trained for a fixed number of epochs which leads to some adequate convergence or until a suitable convergence criterion is satisfied. The respective training procedure is shown in algorithm 1.

Algorithm 1 Generator pre-training algorithm.

```
Require: generator $g$, epochs, training data
 1: Initialize $g$ by Xavier normal initialization
 2: for epoch in epochs do
 3:     for batch in training data do
 4:         Set training sequences $Y_{0:T-1} = (0, y_1, ..., y_{T-1})$
 5:         Set label sequences $Y_{1:T} = (y_1, ..., y_T)$
 6:         Generate sequences $\hat{Y}_{1:T} = (\hat{y}_1, ..., \hat{y}_T) \sim g(Y_{0:T-1})$
 7:         Calculate loss $NLL(\hat{Y}_{1:T}, Y_{1:T})$
 8:         Update generator parameters
 9:     end for
10: end for
```

3.2.4.2 Discriminator Pre-Training

In the next step the discriminator is prepared for the adversarial training. For the pre-training of the discriminator in each case a batch of real sequences and a batch of fake sequences sampled by the already pre-trained generator are provided. To balance the decision process the generator always produces sequences of equal length to the real ones. Then the discriminator tries to decide if a given sequence is fake or real by computing a binary probability vector. Therefore the binary cross-entropy is employed as loss and minimized in the training process. Then the discriminative model is trained for an adequate number of epochs. The initialization again is done by Xavier normal initialization and to prevent exploding gradients I additionally use the gradient norm and gradient clipping techniques. The pseudo-code for the discriminator pre-training can be found in algorithm 2.

Algorithm 2 Discriminator pre-training algorithm.

```
Require: pre-trained generator $g^*$, discriminator $d$, epochs, training data
 1: Initialize $d$ by Xavier normal initialization
 2: for epoch in epochs do
 3:     for batch in training data do
 4:         Get real sequences $Y_{1:T} = (y_1, ..., y_T)$
 5:         Generate fake sequences $\hat{Y}_{1:T} = (\hat{y}_1, ..., \hat{y}_T) \sim g^*(0)$
 6:         Predict fake probabilities $p_0 = d(Y_{1:T})$ and $p_1 = d(\hat{Y}_{1:T})$
 7:         Calculate loss $BCE(p_0, 0) + BCE(p_1, 1)$
 8:         Update discriminator parameters
 9:     end for
10: end for
```

To speed up the training of the algorithm I create a buffer of generator samples that feeds into the discriminator in the same way as the real data is provided, so the sampling step (line 5 in algorithm 2) can be skipped.

3.2.4.3 Adversarial Training

The final and major training stage is the adversarial training where the generator and discriminator are trained in an alternating fashion.

First the generator is trained for one iteration. A batch of sample sequences is generated and for each part of the sequence the ETL is calculated by means of the policy gradient. For the roll-out policy I create a similar generator model g' which then is periodically updated with the parameter values of the original model. The update rate $r_u \in [0, 1]$ controls by how much the parameters are adapted in each iteration. The roll-out model g' completes the given sequences and the final loss is calculated by the discriminator. This procedure is repeated N times to result in a N-time MC search. Then the respective gradient is computed and the weights are updated via Adam.

In the next step the discriminator once again receives real and fake samples and predicts the fake probability. Here the training speed can be improved again by pre-sampling fake sequences produced by the generator and buffering them for efficient use in the loop. However this time the EBGAN structure is employed and so instead of a BCE loss a general margin loss in conjunction with the mean squared error (MSE) is used.

In the final step the simulated annealing of the learning rates for the optimizers is executed and the whole process starts anew. The procedure is repeated until the GAN4Rec converges. The respective training process is presented in algorithm 3.

3.2.5 Programming Framework

As basic programming language for this project I choose python[1] because of its comprehensive standard libraries as well as a plenitude of specialized open-source packages for data

[1]https://www.python.org/

Algorithm 3 Adversarial training algorithm.

```
Require:
    pre-trained generator $g^*$, pre-trained discriminator $d^*$, training data, $r_u$, $\lambda$, $margin$

 1: Create roll-out model $g' \leftarrow g^*$
 2: repeat
 3:     for batch in training data do
 4:         Generate fake sequences $\hat{Y}_{1:T} = (\hat{y}_1, ..., \hat{y}_T) \sim g^*(0)$
 5:         Compute $Q_{\mathcal{L}}(Y_{0:T-1}, \hat{Y}_{1:T}; g') + \lambda \bar{H}^{-1}(p_g)$
 6:         Update generator parameters by policy gradient
 7:     end for
 8:     Update roll-out model: $\theta_{g'} = r_u \theta_{g^*} + (1 - r_u)\theta_{g'}$
 9:     for batch in training data do
10:         Get real sequences $Y_{1:T} = (y_1, ..., y_T)$
11:         Generate fake sequences $\hat{Y}_{1:T} = (\hat{y}_1, ..., \hat{y}_T) \sim g^*(0)$
12:         Predict fake probabilities $p_0 = d^*(Y_{1:T})$ and $p_1 = d^*(\hat{Y}_{1:T})$
13:         Calculate loss $MSE(p_0, 0) + max(0, margin - MSE(p_1, 1))$
14:         Update discriminator parameters
15:     end for
16: until convergence
```

science applications. The core libraries I use are the scientific computing library *numpy*[2], the data analysis and pre-processing library *pandas*[3] and the visualization and plotting library *matplotlib*[4]. Furthermore for the handling of the huge training data I employ the relational database system *SQLite*[5] together with its respective python package *sqlite3*[6].

Basically there are several advanced ecosystems and frameworks for ANN available for python. These include *Caffe*, *Theano*, *TensorFlow*, *Keras* and *Lasagne* as well as *PyTorch* and others. Therefore a lot of more or less comprehensive and up-to-date frameworks for ANN could be chosen. For a large part this decision might not be made because of the actual scope or performance of the framework but rather by personal preference. Nevertheless I use PyTorch[7] for the following general reasons:

- Very intuitive application for experienced python programmers, because of ”pythonic” principles like explicit rather than implicit formulation and simple structures preferred to complex ones,
- Highly integrated with core libraries like numpy,
- Native support for graphics processing units (GPU),
- Several levels of abstraction and smooth transition between those levels,
- Dynamic *define-by-run* framework without compilation step and therefore
  - easy debugging with standard python development debugging tools and

[2]http://www.numpy.org/
[3]https://pandas.pydata.org/
[4]https://matplotlib.org/
[5]https://www.sqlite.org/
[6]https://docs.python.org/3/library/sqlite3
[7]http://pytorch.org/

  - structural and computational advantages when handling RNN and varying length input sequences.

- Furthermore an active and increasing user community and a responsive core development team.

In general PyTorch is a deeply integrated python interface build on top of the Torch computation engine [CBM02] comparable to the Lua frontend. The primary development is done by the artificial intelligence research group of Facebook but it was made publicly available under an open-source license in January 2017.

4. Evaluation

To be able to compare the proposed model to other state-of-the-art approaches a comprehensive evaluation study has to be designed. This involves the selection of one or more datasets that accurately represent the focus problem task, comparable data pre-processing techniques and suitable metrics to measure the model performances. Furthermore proper reference models have to be chosen and comparably implemented.

4.1 Data

The dataset of choice for the session-based recommendation task is the *YOOCHOOSE* dataset that was made available for the *RecSys Challenge 2015*[1]. The same data was used by [Hid+15] to benchmark their model.

The dataset consists of a collection of online shopping sessions on the platform of an European retail store in the year 2014. It is comprised by the clickstreams of users navigating the shop as well as respective buy events where appropriate. For the challenge a training set with complete data and a test set were provided. Following the example of [Hid+15] I only use the clickstream data of the training set in the evaluation.

The data is given in a format in which every record consists of a *Session ID*, a *Timestamp*, an *Item ID* and the affiliation to a particular *Category*. To arrive at the restricted use case I additionally remove the category information and transform the date-time strings into unix timestamps. In this form the original data set consists of 32,995,016 records. A small part of the resulting dataset is shown in table 4.1.

Session ID	Item ID	Time
1	214536502	1396860669.277
1	214536500	1396860849.868
1	214536506	1396860886.998
1	214577561	1396861020.306
2	214662742	1396871797.614
2	214662742	1396871839.373
2	214825110	1396871917.446
2	214757390	1396871990.71
2	214757407	1396872038.247
2	214551617	1396872156.889
3	214716935	1396437466.94
3	214774687	1396437962.515
3	214832672	1396438212.318

Table 4.1: Excerpt of the initial training data.

Then the data is split into a training set and a test set. I follow the example of Hidasi et al. to arrive at a training set that is made up of the sessions of around 6 months containing 32,923,726 events and the test set comprising the 71,290 events of the following day.

[1]http://2015.recsyschallenge.com/challenge.html

In the next step I investigate the distribution of the item occurrences. Initially there are 52703 different items, however most of the clicks belong to a very restricted set of items. This leads to the characteristic *power-law* or *long-tail* distribution that is shown in figure 4.1.

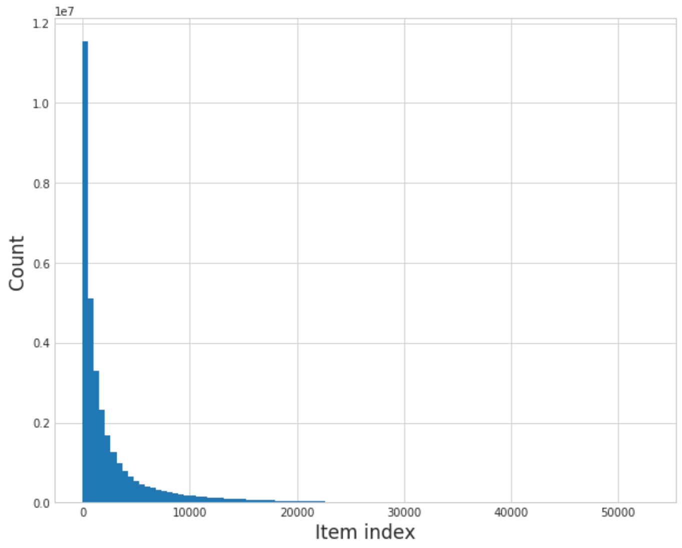

Figure 4.1: Long-tail distribution of the items.

Thus in order to simplify the learning task and to reduce the complexity as well as the computational load I cut the long-tail by restricting the item space to the 70% quantile of the data. That is the part of the data that provides 70% of all clicks, only including clicks on items with a high absolute amount of clicks. This approach results in a dataset which comprises only the 2344 most important items.

However for a tensor processing framework like PyTorch the current format of single event records is not optimal. Therefore I further process the data by combining the records of a particular session to a single clickstream consisting of the chronological sequence of items. Furthermore all times but the start time of the sequence are dropped, resulting in condensed session records of the format shown in table 4.2.

Session ID	**Item ID Sequence**	**StartTime**
1	214536502, 214536500, 214536506, 214577561	1396860669.277
2	214662742, 214662742, 214825110, 214757390, 214757407, 214551617	1396871797.614
3	214716935, 214774687, 214832672	1396437466.94

Table 4.2: Excerpt of the transformed sequence data.

Sequences with a length of 1 are dropped since they don't provide suitable training data to learn sequential behavior. Consequently the resulting item sequences have a minimum length of 2 and maximum length of 200. However exorbitantly large sequence lengths seem to be suspicious. Often such sequences belong to customers that just leave their search in an open browser tab while the online store has no time-out functionality in place and continue their session later on without any causal connection to the previous clickstream. Therefore all sessions with a length of more than 130 are treated as outliers and removed from the dataset. This finally results in a training set comprised by 5,852,490 sessions with an average sequence length of 3.625 whereas the respective median is 3.0. The test set is made up of 6,596 sessions with an according average sequence length of 3.455 and a median of 2.0.

4.2 Metrics

Another important part of the design of a comparative evaluation study is the choice of suitable performance measures to compare the implemented approaches. Fortunately there exist standard measures that are usually employed to compare the model performance in recommendation tasks which involve the generation of a list or respective Top-N recommendation. Most of these measures originate from classical IR tasks. A very important aspect of RS is the *relevance* of the recommended items. To measure this relevance I employ the two following metrics:

- **Recall@N**
  This measure describes how often the recommended item appears in the top-N places of the list of predicted items relative to all relevant items. It is defined as [Dom08]:

$$recall@N = \frac{|\{relevant\ items\} \cap \{predicted\ items\ on\ first\ N\ ranks\}|}{|\{relevant\ items\}|}. \tag{4.1}$$

  Consequently the recall is an *event-based* measure that is independent of the actual rank of the item and just checks if it is amongst the top-N. This case considers certain RS applications where the absolute order of the presented recommendations does not matter. Furthermore the recall was reported to be correlated with the *click-through rate* (CTR) [HT16], an important metric for the online evaluation of RS.

- **MRR@N (Mean Reciprocal Rank)**
  The MRR@N describes the relative appearance of a particular predicted item on each position of the top-N ranks averaged over all computed lists Q [Liu11]; [Cra09]:

$$MRR@N = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{r_i}, \quad \forall r_i > N, \tag{4.2}$$

  where r_i is the ranking position of the relevant item in list i. Therefore this measure is a *query-based* metric that not only sheds light on the correct recommendation, but also if the recommendation was on an appropriate rank within the top-N places of all relevant lists.

The evaluation is done in a next-item prediction setting by providing the item sequence one-by-one and comparing the rank of the prediction to the next item in the sequence.

4.3 Reference Models

To be able to access the comparative performance of the proposed model it needs to be compared to suited baselines and reference models. In this thesis I compare GAN4Rec to three increasingly advanced recommendation approaches.

4.3.1 Popularity Model

The first approach that at the same time represents the bottom baseline is popularity based recommendation. The basic idea of this simple model is to always recommend the most popular items. This approach might seem naive but often represents a strong baseline [Hid+15]. In order to determine the popularity of an item I calculate the *support* of each item on the training data, given for item x by [HGH05]:

$$supp(x) = \frac{|\{s \in S; x \subseteq s\}|}{|S|}, \tag{4.3}$$

where S is the total set of sessions in the training data. Consequently the support is characterized by the relative occurrence of an item compared to all other items. The respective rank of item x is calculated by

$$r_x = \frac{supp(x)}{1 + supp(x)}. \tag{4.4}$$

For the prediction task the top-N highest ranking items are selected and returned as recommendation.

4.3.2 Session-refined Popularity Model

The second approach is superior to the first by additionally considering the occurrence of items seen so far within the same session [Hid+15]. Therefore the calculation of the ranking score changes to:

$$r_x = supp_s(x) + \frac{supp(x)}{1 + supp(x)}, \tag{4.5}$$

where $supp_s(x)$ is the support for item x in session s.

4.3.3 GRU4Rec

The third reference model for the evaluation study is a much more advanced model compared to the first two baselines. GRU4Rec is the state-of-the-art model for session-based recommendation developed by [Hid+15] that was further refined in [HK17]. According to its name this model employs a RNN represented by a GRU. Hidasi et al. propose several loss functions that directly consider the ranking of the predicted items to take care of the special recommendation use case.

- **BPR (Bayesian Personalized Ranking)**
  BPR is a pair-wise ranking loss that compares the score of a particular item to a set of sampled items that are considered negative examples [Ren+09]. The loss is then given by the average of the score differences:

  $$\mathcal{L}_{BPR} = -\frac{1}{n}\sum_{j=1}^{n}\log(\sigma(r - r_j)), \tag{4.6}$$

  where n is the sample size, r is the score of the item and r_j is the score of the negative example j.

- **TOP1**
  The TOP1 loss was especially designed by [Hid+15] and represents a regularized approximation to the relative rank given by

  $$\frac{1}{n}\sum_{j=1}^{n} I\{r_j > r\}, \tag{4.7}$$

  by using a sigmoid to approximate I:

  $$\mathcal{L}_{TOP1} = \frac{1}{n}\sum_{j=1}^{n}\sigma(r_j - r) + \sigma(r_j^2). \tag{4.8}$$

- **BPR-max**
  In their subsequent paper [HK17] the authors propose a refined version of their previous loss functions to overcome vanishing gradient problems encountered when using a huge number of negative samples. The resulting loss is a list-wise ranking loss based on the idea to compare the score of a particular item only to the score of the most relevant negative example. The advanced loss was derived for BPR as well as for TOP1, however the BPR-max loss usually performed better in their experiments and therefore is used in the reference model of this thesis as well:

  $$\mathcal{L}_{BPR-max} = -\log\sum_{j=1}^{n} s_j\sigma(r - r_j), \tag{4.9}$$

  where s_j is the respective softmax score by which the non-differentiable *max* function is approximated.

As an efficient method to provide the negative samples to the loss function the authors propose to use the other examples in the mini-batch as negative examples. This approach is inherently popularity-based since more popular items appear more often in the respective mini-batches. Popularity-based sampling is a reasonable approach since in RS the implicit feedback of *not choosing* an item is much more informative for popular items [Hid+15]. Furthermore in [HK17] an additional set of samples was proposed that is independent of the mini-batches and which was reported to further improve the over-all performance of the model by providing a greater amount of possibly relevant negative examples. The final hyperparameters are described in table 4.3.

Parameter	**Value**
Embedding dim	32
Dropout prob embedding	0.0
Hidden dim	128
Dropout prob hidden	0.35
Num GRU layers	1
Num Linear layers	2
Batch size	64
Num epochs	10
Learning rate	0.01

Table 4.3: Final hyperparameters of GRU4Rec.

4.4 Implementation Challenges and Limitations

In comparison to the challenges w.r.t. the design of the GAN4Rec model that mostly could be resolved by employing the techniques and ideas discussed in 3.2.2, the implementation of a fully functional prototype that works on real data presents additional challenges that sometimes cannot or can only partly be resolved. These challenges in turn most often lead to some limitations of the model and the evaluation setting. The resulting implications are described in the following paragraph.

4.4.1 GAN4Rec Limitations

1. **Hyperparameter tuning**
   As was already mentioned in 2.3.1.7 the search for suitable hyperparameters can easily lead to a combinatorial explosion when the parameter space is sufficiently large. In case of the GAN4Rec this is the case since the model is very complex and consists of two ANN with independent parameterization as well as a plenitude of parameters for loss functions, regularization and general hyperparameters guiding the training procedure. All in all more than 30 parameters need to be set. Even if each parameter was just a binary decision (what they are mostly not) this would lead to 2^{30} possible combinations which cannot be covered by any computational means. Furthermore my experimental setting is restricted to a single machine which is training a model in an average time of 15h so there is no chance to even slightly explore the respective parameter space. For this reason a set of reasonable start parameters was selected from the appropriate scientific papers and then further fine tuned manually following a greedy strategy. Consequently it is almost certain that the final parameterization is not representing the full power of the model.

2. **Generator sampling**
   Since the sampling of a full batch of very long sequences in the roll-out policy in generator training consumes huge amounts of memory and computational resources,

the maximum length of sampled sequences in this special case is set to 40. However the sampling in discriminator training as well as for evaluation is not restricted.

3. **Number of training samples**
   The RL approach involving the gradient approximation via roll-out policy takes a lot of time. A full training run for the generator on the whole data set takes several hours. Although the discriminator is about 4 times faster, the enormous size of the dataset leads to large training times. However the convergence of a GAN depends heavily on the interaction between the generative and the discriminative model in the adversarial training stage with alternating model training. Therefore it is not feasible to fully train each model on the whole dataset since in this case an alternating training iteration would take a tremendous amount of time. Furthermore this could lead to one model overpowering the other since it receives massive training input while the other part of the model is frozen, finally deteriorating convergence. Instead a suitable number of training samples is randomly selected in each iteration and provided as training data for the generator and discriminator.

4.4.2 GRU4Rec Limitations

All the following limitations w.r.t the GRU4Rec model are due to its implementation in the PyTorch framework. The original model is implemented in Theano and for this reason the results reported in the paper cannot directly be compared to the GAN4Rec model. However the re-implementation of GRU4Rec in PyTorch posed several challenges that could not all be resolved completely. On the one hand this is because no implementation details are available in the published papers and on the other hand since the respective Theano code is completely uncommented.

1. **Session parallel mini-batches**
   The data processing in the original model is based on session parallel mini-batches, which means that the item sequences are provided in parallel and if one sequence ends it is replaced by the next available sequence. The reason to do this in Theano is to prevent the RNN from idle states where only a small fragment of the net is effectively used when processing padded sequences and therefore speed up training. However PyTorch is a dynamic framework that has a buffered data feed into the RNN which leads to fast parallel processing of padded sequences. For this reason I do not see the need to implement the processing based on session parallel mini-batches which further complicates the training routines as well as the general structure.

2. **Data format**
   The authors report that they experimented with embedded data as well as data in 1-hot-encoding and found that the embeddings performed slightly worse. However the difference was only marginal which is why I still prefer an embedding approach over the 1-hot-encoding since it comes with additional advantages from the implementation perspective in PyTorch.

3. **Additional sampling**
   As described in 4.3.3 the refined original GRU4Rec model uses additional samples

to increase the amount of available negative examples. However I wasn't able to implement this functionality since it lead to problems when backpropagating the gradient through the computational graph that I couldn't resolve in the given time.

4. **Hyperparameter tuning**
   Since the altered structure does not necessarily work out of the box with the same parameters proposed in the respective papers the same problem concerning hyperparameters that was explained w.r.t. GAN4Rec in the paragraph above might apply. However GRU4Rec has only around 9 parameters to tune and therefore the problem is not that severe and can be fairly overcome with some effort put into a greedy search strategy starting from the proposed parameters.

4.4.3 General Challenges

Finally the massive amount of data made it impossible to pre-process and format it in the RAM so a persistent database had to be used, including the implementation of the respective pre-processing steps in SQLite.

4.5 Discussion of Results

For a comprehensive comparison of the performance of GAN4Rec and the baselines all models are trained on the available training set. In training and in testing all models are provided with fully pre-processed data.

Furthermore for the GRU4Rec and GAN4Rec model a hyperparameter search following the description of 4.4 is executed. Therefore the original training set is further split into a training and a validation set which serves as data for parameter evaluation. The resulting training set consists of 5,846,869 sessions and the validation set is made up of 5,621 sessions. Then the best performing parameters are selected and the respective models are re-trained on the full training set. The results of the model performance evaluated on the hold-out test set are presented in table 4.4. The best results are printed in bold respectively.

Model	**Recall@20**	**MRR@20**
POP	0.0237	0.0118
SPOP	0.2479	0.1239
GRU4Rec	**0.7788**	**0.3579**
GAN4Rec	0.1759	0.0386

Table 4.4: Resulting model performance measured by Recall@20 and MRR@20.

To achieve a meaningful assessment of the performance of the two advanced models whose results can significantly depend on the random values created in initialization I averaged the results of three runs with different seeds for the internal random number generators. Furthermore to get a more comprehensive impression of the performance of the models, table 4.5 summarizes the time necessary for the training of the respective model and the computation of the predictions on the whole test set.

Model	Training time	Prediction time
POP	**0.4**	**0.0588**
SPOP	**0.4**	0.0803
GRU4Rec	28671.6	0.2546
GAN4Rec	40352.7	0.2061

Table 4.5: Respective training and prediction times of the models in seconds.

As becomes apparent from table 4.4, the GRU4Rec model achieves by far the most superior performance on the next-item prediction task measured by Recall@20 as well as MRR@20. With a recall of 0.7788 and a MRR of 0.3579 its improvements w.r.t. the second best model (SPOP) are around 214% and 189% respectively. The worst result belongs to the POP model. This result is not surprising since the naive approach of the POP model is based only on the general item support in the training set. However much more surprising is that the SPOP model is able to achieve results that are more than ten times better. Its performance is also superior to the GAN4Rec model developed in this thesis.

If we take another look on the data, this can be partly explained by an important characteristic of the clickstreams from the YOOCHOOSE dataset. Recall that the data consists of clickstreams representing the consecutive views of a customer in a particular session. Therefore an item that was bought in the end, most certainly has been viewed last in the preceding click sequence. Furthermore often the final item is not just viewed last at the end of the search and then instantly bought, but rather has been already encountered at least once before in the sequence. Keeping this in mind and regarding the test set with an average sequence length of 3.455 and a median of 2.0 the chance of an item appearing multiple times in a short sequence is pretty high. Although not all sequences lead to a purchase event, this property has a huge impact on the prediction capability and performance of the SPOP model.

Apparently GAN4Rec is not able to fully capture the characteristics of the data and therefore produces weak results even compared to the quite simple SPOP model. Although the model converges, it seems to converge to a quite suboptimal equilibrium. With further examination of GAN4Rec this could be attributed to its architecture that might be overly complex for the problem. Another reason might be the training procedure for the generator. The RL approach with MC search and policy gradient aims to find full-length sequences that resemble the real data. The implicit notion of this approach is to allow for mistakes or weak decisions within the sequence as long as they lead to a full sequence with a higher probability to be labeled as real. This might be an preferable behavior in tasks that depend on the whole sequence or only its last element for evaluation. However the model is evaluated in a scenario that focuses on an element-wise measurement of quality, namely the prediction of the following next item for each item in the sequence. For this reason the model suffers from low evaluation scores in this particular recommendation task. Furthermore, directly associated with the complexity of the model the enormous hyperparameter space is a huge problem for the model optimization and might be heavily responsible for the weak results.

Regarding the respective times needed to train the different models, with around 0.4 seconds the necessary duration for the training of the simple models is hardly even comparable to the time needed to train GRU4Rec and GAN4Rec. Nevertheless this is a characteristic that is trivial when comparing the different complexity and structure of the models and its training procedures. However when examining the times for prediction the differences are less obvious. Here we compare 0.0588 seconds and 0.0803 seconds for the simple models with respectively 0.2546 seconds and 0.2061 seconds for GRU4Rec and GAN4Rec. If we recall that in practice, at least when assuming that the model is not retrained on a daily basis, the time needed to compute predictions is the major time-based indicator for the usability of the RS, then the advanced models are principally on the same page as the simple ones.

The training time for the GAN4Rec model consists of an average time for generator pre-training of 5591 seconds, an average time for discriminator pre-training of 5643.5 seconds and average adversarial training time of 29118.2 seconds, accumulating to a total of 40352.7 seconds, what roughly equals 11 hours. Although I could compare the training time for the advanced models, this makes no inherent sense since the performance of the models is totally different. Therefore the GAN4Rec model might achieve at least comparable results when just trained for half of the time since this might not have a strong and immediate effect on its suboptimal ability of capturing the characteristics of the data, while the performance of GRU4Rec could decrease significantly.

5. Conclusion

In this thesis I investigated the utility of a RS based on the GAN architecture for recommendation tasks on sequential data. Basically the contribution consists of three major parts.

Starting from the ideas presented by Yoo et al. [Yoo+17] I developed and implemented a working prototype of the proposed model. Therefore design decisions for model structure, loss functions, initialization and optimization routines as well as the training procedure were made. The final model employs two RNN in a pre-training and adversarial training stage.

Moreover I refined the model architecture with a new regularization term called *normalized inverse entropy* which, compared to the negative entropy, represents a non-negative term that can be directly added to the training objective of the model.

Finally I carried out an almost complete transfer of the whole GRU4Rec model of Hidasi et al. [Hid+15] and its training procedures from the Theano to the PyTorch computational framework for the purpose of objective performance comparison. This included the thorough annotation and explanation of the respective code which was not available in Theano.

In the subsequent comparative evaluation study I set GAN4Rec against three increasingly complex reference models. The results confirm the capacity and performance of the state-of-the-art GRU4Rec model while the application of the GAN architecture, at least as it was presented in this thesis, can not be fully justified.

Recalling the two initial research questions, I showed how to design a GAN-based RS that performs better than random and also beats the simple POP model. However GAN4Rec could not surpass the second baseline SPOP and compared to the state-of-the-art represented by GRU4Rec the proposed model shows significantly lower performance.

Therefore the proposition that the GAN architecture is able to improve recommendation relevance for session-based recommendation over the state-of-the-art can not be supported by this thesis.

6. Outlook

The GAN4Rec model proposed in this thesis is not able to deliver the expected results in the respective recommendation task. However there is still a bunch of things that I didn't try and implement to improve the performance of the status quo.

First of all I expect that a thorough re-examination of the separate hyperparameters and their influence on the training process will very likely improve the performance significantly. However this kind of search and investigation was not possible for me within the computational and time constraints of this masters thesis.

Additional conclusions could be drawn from a different recommendation task in the comparative study, focused on long sequences, that are evaluated as whole, e.g. automatic play-list completion as was proposed in [Yoo+17]. The GAN4Rec structure and its training procedure might be more suited for such a scenario.

A quite different approach could be the use of the very recently discussed deployment of CNN with an (hierarchical) attention mechanism [VDO+16] that achieves surprising results on sequential data and outperforms RNN in several tasks [BKK18]. The application of such ANN is a very interesting topic and could lead to further evolvement of the model.

Although the results of the GAN model in this thesis are rather weak, there are also other applications that could shift the focus of RS on GAN. While privacy issues are not a severe pain-point for session-only data, it is much more so for traditional RS involving user profiles and lots of private user information. Naturally more information and therefore more dense training data in most cases leads to better models and finally improved recommendations in a RS. However in such scenarios the data privacy is a very delicate topic. A possible solution could again be given by special GAN architectures that were shown to be able to produce differentially private results for different ML tasks [Xie+18]; [ZJW18].

Therefore besides the possible improvement and refinement of GAN4Rec other RS areas with application of GAN are likely to arise in the next years.

Bibliography

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases". In: *Acm sigmod record*. Vol. 22. 2. ACM. 1993, pp. 207–216.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". In: *IEEE transactions on knowledge and data engineering* 17.6 (2005), pp. 734–749.

[Aïm+08] Esma Aïmeur et al. "Alambic: a privacy-preserving recommender system for electronic commerce". In: *International Journal of Information Security* 7.5 (2008), pp. 307–334. ISSN: 1615-5270. DOI: `10.1007/s10207-007-0049-3`. URL: `https://doi.org/10.1007/s10207-007-0049-3`.

[BB12] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[Bel61] Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 1961.

[Ber+11] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*. 2011, pp. 2546–2554.

[Bis16] Christopher M. Bishop. *PATTERN RECOGNITION AND MACHINE LEARNING*. Springer-Verlag New York, 2016.

[BKK18] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

[Bob+13] Jesús Bobadilla et al. "Recommender systems survey". In: *Knowledge-based systems* 46 (2013), pp. 109–132.

[Bow09] Samuel Bowles. *Microeconomics: behavior, institutions, and evolution*. Princeton University Press, 2009.

[Bro+12] Cameron B Browne et al. "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Cao+09] Huanhuan Cao et al. “Towards context-aware search by learning a very large variable length hidden markov model from search logs”. In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 191–200.

[Car14] Tom Carter. *An introduction to information theory and entropy*. 2014. URL: `http://csustan.csustan.edu/~tom/Lecture-Notes/Information-Theory/info-lec.pdf` (visited on 03/19/2018).

[CBM02] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. *Torch: a modular machine learning software library*. Tech. rep. Idiap, 2002.

[CCB11] Iván Cantador, Pablo Castells, and Alejandro Bellogín. “An enhanced semantic layer for hybrid recommender systems: Application to news recommendation”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 7.1 (2011), pp. 44–78.

[CDM15] Marc Claesen and Bart De Moor. “Hyperparameter search in machine learning”. In: *arXiv preprint arXiv:1502.02127* (2015).

[Cho+14a] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[Cho+14b] Kyunghyun Cho et al. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).

[Cra09] Nick Craswell. “Mean reciprocal rank”. In: *Encyclopedia of Database Systems*. Springer, 2009, pp. 1703–1703.

[CWY04] Pei-Yu Chen, Shin-yi Wu, and Jungsun Yoon. “The impact of online recommendations and consumer feedback on sales”. In: *ICIS 2004 Proceedings* (2004), p. 58.

[Dai+16] Hanjun Dai et al. “Recurrent coevolutionary latent feature processes for continuous-time recommendation”. In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM. 2016, pp. 29–34.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[Dia+08] M Benjamin Dias et al. “The value of personalised recommender systems to e-business: a case study”. In: *Proceedings of the 2008 ACM conference on Recommender systems*. ACM. 2008, pp. 291–294.

[Die02] Thomas G. Dietterich. “Machine Learning for Sequential Data: A Review”. In: *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*. Ed. by Terry Caelli et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 15–30. ISBN: 978-3-540-70659-5. DOI: `10.1007/3-540-70659-3_2`. URL: `https://doi.org/10.1007/3-540-70659-3_2`.

[Dom08] Sándor Dominich. *The modern algebra of information retrieval.* Springer, 2008.

[Fin+16] Chelsea Finn et al. "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models". In: *arXiv preprint arXiv:1611.03852* (2016).

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics.* 2010, pp. 249–256.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[Gel+12] Sylvain Gelly et al. "The grand challenge of computer Go: Monte Carlo tree search and extensions". In: *Communications of the ACM* 55.3 (2012), pp. 106–113.

[Goo+14] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems.* 2014, pp. 2672–2680.

[Goo14] Ian J Goodfellow. "On distinguishability criteria for estimating generative models". In: *arXiv preprint arXiv:1412.6515* (2014).

[Goo16a] Ian Goodfellow. "NIPS 2016 tutorial: Generative adversarial networks". In: *arXiv preprint arXiv:1701.00160* (2016).

[Goo16b] Ian J Goodfellow. *Generative Adversarial Networks for Text.* 2016. URL: `https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/` (visited on 03/17/2018).

[Gra12] Alex Graves. "Supervised sequence labelling". In: *Supervised sequence labelling with recurrent neural networks.* Springer, 2012, pp. 5–13.

[GSC99] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[GSS02] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143.

[HGH05] Michael Hahsler, Bettina Grün, and Kurt Hornik. "A computational environment for mining association rules and frequent item sets". In: (2005).

[Hid+15] Balázs Hidasi et al. "Session-based recommendations with recurrent neural networks". In: *arXiv preprint arXiv:1511.06939* (2015).

[Hid+16] Balázs Hidasi et al. "Parallel recurrent neural network architectures for feature-rich session-based recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems.* ACM. 2016, pp. 241–248.

[Hin+12] Geoffrey Hinton et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[HK17] Balázs Hidasi and Alexandros Karatzoglou. “Recurrent Neural Networks with Top-k Gains for Session-based Recommendations”. In: *arXiv preprint arXiv:1706.03847* (2017).

[Hop82] John J Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[HT16] Balázs Hidasi and Domonkos Tikk. “General factorization framework for context-aware recommendations”. In: *Data Mining and Knowledge Discovery* 30.2 (2016), pp. 342–371.

[Im+16] Daniel Jiwoong Im et al. “Generating images with recurrent adversarial networks”. In: *arXiv preprint arXiv:1602.05110* (2016).

[Jam+13] Gareth James et al. *An introduction to statistical learning.* Vol. 112. Springer, 2013.

[Ji+16] Youchun Ji et al. “Regularized singular value decomposition in news recommendation system”. In: *Computer Science & Education (ICCSE), 2016 11th International Conference on.* IEEE. 2016, pp. 621–626.

[Jol02] Ian T Jolliffe. “Principal component analysis and factor analysis”. In: *Principal component analysis* (2002), pp. 150–166.

[KB14] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009).

[KGB14] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A convolutional neural network for modelling sentences”. In: *arXiv preprint arXiv:1404.2188* (2014).

[Koh82] Teuvo Kohonen. “Self-organized formation of topologically correct feature maps”. In: *Biological cybernetics* 43.1 (1982), pp. 59–69.

[KW13] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[LDP10] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. “Personalized news recommendation based on click behavior”. In: *Proceedings of the 15th international conference on Intelligent user interfaces.* ACM. 2010, pp. 31–40.

[LeC+06] Yann LeCun et al. “A tutorial on energy-based learning”. In: *Predicting structured data* 1.0 (2006).

[LFR06] Shyong Lam, Dan Frankowski, and John Riedl. “Do you trust your recommendations? An exploration of security and privacy issues in recommender systems”. In: *Emerging trends in information and communication security* (2006), pp. 14–29.

[Liu11] Tie-Yan Liu. *Learning to rank for information retrieval.* Springer Science & Business Media, 2011.

[Lü+12] Linyuan Lü et al. “Recommender systems”. In: *Physics Reports* 519.1 (2012), pp. 1–49.

[Mer+16] Paul Merolla et al. “Deep neural networks are robust to weight binarization and other non-linear distortions”. In: *arXiv preprint arXiv:1606.01981* (2016).

[Met+16] Luke Metz et al. “Unrolled generative adversarial networks”. In: *arXiv preprint arXiv:1611.02163* (2016).

[MH09] Andriy Mnih and Geoffrey E Hinton. “A scalable hierarchical distributed language model”. In: *Advances in neural information processing systems.* 2009, pp. 1081–1088.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. ICML.* Vol. 30. 1. 2013.

[Mik+10] Tomáš Mikolov et al. “Recurrent neural network based language model”. In: *Eleventh Annual Conference of the International Speech Communication Association.* 2010.

[Mik+13] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems.* 2013, pp. 3111–3119.

[Mil07] Elizabeth Million. “The hadamard product”. In: *Course Notes* 3 (2007), p. 6.

[Mit+97] Tom M Mitchell et al. *Machine learning. WCB.* 1997. URL: `"http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html"`.

[NH10] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10).* 2010, pp. 807–814.

[NIH97] Ury Naftaly, Nathan Intrator, and David Horn. “Optimal ensemble averaging of neural networks”. In: *Network: Computation in Neural Systems* 8.3 (1997), pp. 283–296.

[OR94] Martin J Osborne and Ariel Rubinstein. *A course in game theory.* MIT press, 1994.

[Par11] Eli Pariser. *The filter bubble: What the Internet is hiding from you.* Penguin UK, 2011.

[Pef+06] Ken Peffers et al. “The design science research process: a model for producing and presenting information systems research”. In: *Proceedings of the first international conference on design science research in information systems and technology (DESRIST 2006).* sn. 2006, pp. 83–106.

[PH15] Andre Luiz Vizine Pereira and Eduardo Raul Hruschka. “Simultaneous co-clustering and learning to address the cold start problem in recommender systems”. In: *Knowledge-Based Systems* 82 (2015), pp. 11–19.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International Conference on Machine Learning.* 2013, pp. 1310–1318.

[Qua+17] Massimo Quadrana et al. “Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks”. In: *arXiv preprint arXiv:1706.04148* (2017).

[Ren+09] Steffen Rendle et al. “BPR: Bayesian personalized ranking from implicit feedback”. In: *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence.* AUAI Press. 2009, pp. 452–461.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), p. 533.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[Ros+04] Lorenzo Rosasco et al. “Are loss functions all the same?” In: *Neural Computation* 16.5 (2004), pp. 1063–1076.

[Ros58] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.

[Rus+15] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* Vol. 1. 1. MIT press Cambridge, 1998.

[Sha01] Claude Elwood Shannon. “A mathematical theory of communication”. In: *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001), pp. 3–55.

[SKR99] J Ben Schafer, Joseph Konstan, and John Riedl. “Recommender systems in e-commerce”. In: *Proceedings of the 1st ACM conference on Electronic commerce.* ACM. 1999, pp. 158–166.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems.* 2012, pp. 2951–2959.

[SLH14] Yue Shi, Martha Larson, and Alan Hanjalic. “Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges”. In: *ACM Computing Surveys (CSUR)* 47.1 (2014), p. 3.

[SMH07] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. “Restricted Boltzmann machines for collaborative filtering”. In: *Proceedings of the 24th international conference on Machine learning.* ACM. 2007, pp. 791–798.

[SP97] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[Sri+14] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[SSM12] Nachiketa Sahoo, Param Vir Singh, and Tridas Mukhopadhyay. “A hidden Markov model for collaborative filtering”. In: *Mis Quarterly* (2012), pp. 1329–1356.

[SV17] Elena Smirnova and Flavian Vasile. “Contextual Sequence Modeling for Recommendation with Recurrent Neural Networks”. In: *arXiv preprint arXiv:1706.07684* (2017).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems.* 2014, pp. 3104–3112.

[TH12] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[Tik43] Andrey Nikolayevich Tikhonov. “On the stability of inverse problems”. In: *Dokl. Akad. Nauk SSSR.* Vol. 39. 1943, pp. 195–198.

[TXL16] Yong Kiam Tan, Xinxing Xu, and Yong Liu. “Improved recurrent neural networks for session-based recommendations”. In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems.* ACM. 2016, pp. 17–22.

[Vap99] Vladimir Naumovich Vapnik. “An overview of statistical learning theory”. In: *IEEE transactions on neural networks* 10.5 (1999), pp. 988–999.

[VDO+16] Aaron Van Den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

[Wes14] Peter H Westfall. “Kurtosis as peakedness, 1905–2014. RIP”. In: *The American Statistician* 68.3 (2014), pp. 191–195.

[Wil92] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Reinforcement Learning.* Springer, 1992, pp. 5–32.

[WM97] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[WR05] Scott A Wymer and Elizabeth A Regan. “Factors influencing e-commerce adoption and use by small and medium businesses”. In: *Electronic markets* 15.4 (2005), pp. 438–453.

[WWY15] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. “Collaborative deep learning for recommender systems”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM. 2015, pp. 1235–1244.

[Xie+18] Liyang Xie et al. “Differentially Private Generative Adversarial Network”. In: *arXiv preprint arXiv:1802.06739* (2018).

[Yan+17] Zhen Yang et al. “Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets”. In: *arXiv preprint arXiv:1703.04887* (2017).

[Yoo+17] Jaeyoon Yoo et al. “Energy-Based Sequence GANs for Recommendation and Their Connection to Imitation Learning”. In: *arXiv preprint arXiv:1706.09200* (2017).

[YRC07] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. “On early stopping in gradient descent learning”. In: *Constructive Approximation* 26.2 (2007), pp. 289–315.

[Yu+17] Lantao Yu et al. “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.” In: *AAAI.* 2017, pp. 2852–2858.

[Zei12] Matthew D Zeiler. “ADADELTA: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[Zha+10] Justin Zhan et al. “Privacy-preserving collaborative recommender systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.4 (2010), pp. 472–476.

[Zie+08] Brian D Ziebart et al. “Maximum Entropy Inverse Reinforcement Learning.” In: *AAAI.* Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[ZJW18] Xinyang Zhang, Shouling Ji, and Ting Wang. “Differentially Private Releasing via Deep Generative Model”. In: *arXiv preprint arXiv:1801.01594* (2018).

[ZML16] Junbo Zhao, Michael Mathieu, and Yann LeCun. “Energy-based generative adversarial network”. In: *arXiv preprint arXiv:1609.03126* (2016).

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 9. Mai 2018

- - - - - - - - - - - - - - - - -

(Jonas Falkner)