Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis Nr. 265

Systems Group, Department of Computer Science, ETH Zurich

in collaboration with

Digitec Galaxus AG

Applying the GAN Framework to Recommender Systems

by

Mohammed Ajil

Supervised by

Bojan Karlas, Ce Zhang

September 13, 2019

inf | Informatik
Computer Science

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Applying the GAN Framework to Recommender Systems

Master Thesis

Mohammed Ajil

September 13, 2019

Advisors: Prof. Ce Zhang, Bojan Karlas
Department of Computer Science, ETH Zürich

Abstract

This example thesis briefly shows the main features of our thesis style, and how to use it for your purposes.

Contents

Chapter 1

Introduction

- Recommender System are a field in machine learning that get more and more attention
- In principle the goal of a recommender system is to recommend items (in the case of an e-commerce system products) to users
- The idea is that the system helps the user navigate the mass of products available and to show the user what is relevant
- A recomender system can be tuned to optimize for different metrics such as click through rate, conversion rate etc.
- In the past there have been mainly to approaches to recommender systems user based and item based
- In the last years there have been more and more proposals for session based approaches (refer to paper that does the survey)
- Specifically GRU4Rec has gained attention as a RNN based approach to solving this problem
- In a session based setting we try to model the sequence of events a user makes when he browses the product catalog instead of items or users themselves.
- This has the following advantages: Usually we have a lot of users that only visit the site once or twice a year, these users are very difficult to model. The same goes for products, there are a lot of products that have limited attention from the userbase, therefore it is difficult to get a useful representation for products like this.
- In a session based setting we model the sequence of events, which is more useful since there is a lot more data, and sessions are comparable even if we do not know which user is online.

- Session based approaches work in both settings where we dont know the user and where we know the user
- Item based are for unkown users and user based for known users
- The goal of the thesis is twofold:
  - First we want to explore the capabilities of such a system in a production setting with real world data. Are we really better than simple approaches like "often bought together"?
  - Also we want to compare it to a more sophisticated system that is a blackbox and consumed as a service provided by Google.
  - Third, since this is an RNN approach we want to find out if we can improve the system by applying the GAN framework in form of professor forcing
  - Fourth, the system inherently produces embeddings for products and users, we want to find out if we can improve the overall performance if we use a pretrained product embedding which captures the semantic similarity of products

Chapter 2

Background

2.1 Recommendation Systems

2.1.1 Problem Statement

Generally speaking recommendation systems are concerned with recommending items to be of use to users. The recommendations should help the users decide which items to buy, music to listen to, what news articles to read etc. Virtually any decision-process can be made easier for users by providing recommendations. Typically recommendation systems are used by online services, famously for example by Spotify for music [2] and YouTube for videos [3]. By using these online service the users generate data that the service can use to improve the recommendations, for example rating videos gives the operator of the service a datapoint on how much a video is liked by a specific user. However also retailers are known to use recommendation systems, based on data generated by customer loyalty programs such as Migros Cumulus [7] retailers tailor coupons or other offers to their customers. In principle data of users interacting with items (viewing, buying, rating etc.) serves as the basis for a recommendation system. Depending on the use-case this data is utilized to assign scores to items depending on the user. The semantic meaning of a score depends on the objective of a recommendation system. For example a system which recommends products to be bought might assign a "probability of purchase", whereas a system which recommends videos might predict the probability of a user watching a video to the end.
Bringing this all together we can define a general recommendation system with the following function:

$$s = f(i, u, h_u) \tag{2.1}$$

Where s is the assigned score to item i for the user u and the users history u_h. The users history represents all the previous interactions with items.

Essentially when we design a recommendation engine we want to learn the function f. To assess the quality of the learned function we need to know the "true" score for the specific item and user, this can be done in different ways. Usually during training we will hold off later interactions with items, and test the learned functions on those. However as soon as the users sees recommendations, we essentially change the reality, i.e. we don't know what the user saw as recommendations, if any, in the user history, therefore it makes sense to also assess the performance of a recommendation engine in production.

2.1.2 Properties of Recommendation Systems

In the following we will look at different properties of recommendation systems. Typically recommendation systems have a combination of the following properties.

User-based User-based recommendation system base the information used to make recommendations mainly on properties of the user, such as gender, age, etc. Also information from the history of the user can be used, for example what the user has bought or read before. Essentially that means when we try to generate recommendations for a specific user we try to find similar users and recommend items that the similar users liked.

Item-based Item-based recommendation system use mostly information about the item to produce recommendations, such as item type (e.g. genre of a movie). However as the name suggests, the basis for a recommendation is always a specific item, for example a product a user is looking at online. This item the user is currently interacting with is referred to as the "active item". Usually item-based recommendation systems then try to find similar items to the currently active item. The method of finding similar items can be arbitrarily complex, identifying similar items is its own field of research. What is also often done is combine this approach with a user-based component, where the similar items are sorted or filtered based on the user. An example of this might be the following:

- Steve is a male looking at black shirts.
- When extracting similar products we find a range of black shirts, also containing womens shirts.
- If we would recommend items by popularity the womens shirts would appear as the first recommendation, since women typically buy more shirts.
- Instead of directly displaying the recommendations we filter out the womens shirts that is found in this selection.

Session-based Session-based recommendation systems are a rather new form of such a system. A session is a sequence of interactions from a user with one or several items. Depending on the use-case and setting this can be defined differently. Usually online-services define a session as the sequence of interactions a user produces on the site until closing the browser window. Obviously sessions can have variable lengths, therefore it is difficult to directly feed that information into a recommendation system. However with the rise of Recurrent Neural Networks (c.f. 2.2.1) a powerful tool for handling variable length sequence data becomes available. Session-based recommendation systems use RNNs to model the sequence data generated by sessions to achieve two things. First by using RNNs we can extract a fixed dimensional representation of a specific session, this allows us to compare different sessions. Second by using the fixed representation of a session we can try to identify the intent a user has in a specific session, based on this recommendations can be made to fulfill the users intent. An intent can be defined as the goal the user has in a specific session. In the example of an online-shop there are a few different, well-known intents identified by analysts such as: Browsing for inspiration, searching for a specific product, buying a specific product, researching products etc. Also these intents exist in different contexts, in the case of an online-shop the contexts can be different product types (mobile phone, couch, dining table) etc. From the above explanation it is intuitive to see why these systems are more desired by operators of online services, since the recommendations can be targeted much more specific to the user and his intent, instead of just general information of the user and the active item.

Collaborative Collaborative recommendation-systems mostly use interaction data to generate recommendations. For example we use the clicks on products as a data source and then predict which products the user will click next. However the collaborative aspect comes from the fact that we source other users interactions as a basis for the recommendations. In principle we view different users as versions of possible behaviour of a user, the more interactions two users have in common, the more similar they are assumed to be. Therefore we can extend the behaviour (i.e. product views) of a user by looking at what similar users have done on the same active item.

Content-based In contrast to collaborative recommendation systems, content-based systems heavily rely on so called content information. Content information refers to the actual content of different items. The name stems from early recommendation systems which mainly focussed on recommending media. The idea is to actually analyze the content of an item, be it the images in a movie, the soundwaves of a song or the text in a book. The assumption is that a user likes to see items that are similar to each other

(e.g. a user who mostly likes action films). However this method can also be used in different contexts, such as online-shopping, where the "content" of an item might be its textual description or an image of the actual product.

2.1.3 Well-Known Examples

In the following sections we will look at some well-known examples of recommendation engines and their properties.

Collaborative Filtering via Matrix Factorization

As the name suggests model is a collaborative approach to a recommendation engine, meaning the model uses primarily interaction data between users and items. As mentioned in [6] the main idea behind collaborative filtering is to recommend items to the user based on what users with similar taste liked in the past. Collaborative filtering refers to a class of models that use similar users' tastes to recommend items. Also this class of models uses the user as a basis for recommendations and not the active item, making it a user-based recommendation system. As seen in [1] there are many ways of implementing a collaborative filtering system. However the important part is the problem representation.

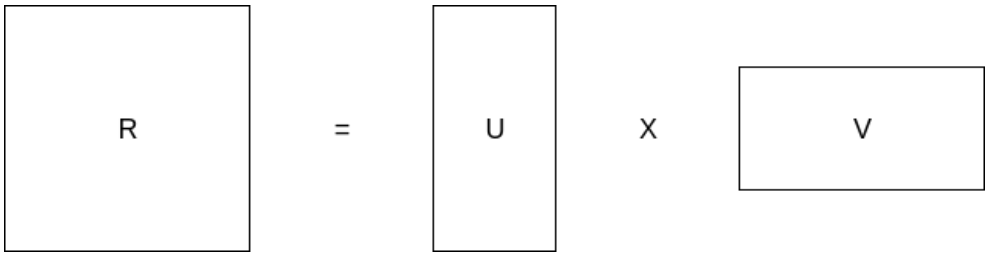

Figure 2.1: Matrix Completion via Matrix Factorization

In the Model-Based approach the problem is represented as a matrix completion problem. The matrix R in figure 2.1 represents the interactions of users and items, where u_{ij} represents the interaction of user i with item j. Usually when dealing with such interaction data, this matrix is very sparse, since in general a user interacts with a small set out of possibly millions of items. The main idea behind this approach is to fill in the missing entries of the matrix. To achieve that we define two randomly initialized matrices U and V which when multiplied produce a matrix of the same shape as U. As explained in [1] these two matrices represent low rank representations of users and items respectively. The next step is to use a optimization algorithm to fit U and V such that $r_{ij} \approx UV_{ij}$. The error of the chosen optimization algorithm however is only applied to entries of R which are known. Therefore

when we find U and V such that the values for the known entries match the values in R we assume that we also found a good approximation for the values unkown in R and use these to predict the interaction of the respective users and items.

Often Bought Together

Often/frequently bought together is a recommendation system very popular in online-stores. The idea behind it is to recommend products that complement the one the user is intending to buy. A classical example for this would be to recommend a protection case when the users adds a smartphone to the basket. Therefore it is a item-based approach. The implementation of this recommendation system can be done in a rather simple way, but can be improved a lot by complex systems, which could personalize the recommendations by using the users history, thereby extending it with a user-based component. However the basic idea stays the same, as the name suggests, finding items that are frequently bought together. The simplest, yet still effective, implementation of this is to just count how many times products appear in the same order as other products, i.e. for each combination of two products i and j we will have a count c_{ij} of how many times these products were bought together. When a user adds a product to the basket, we extract the products with the highest count from a database and recommend these to users.

2.2 Concepts

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks or RNNs are a form of artificial neural networks, which allow to explicitly model sequence input data. The enabling factor for this is that RNNs allow the output of the network to be fed back in.

Further RNNs carry a so called hidden state, which is propagated along the temporal axis. The following two equations from [4] describe the general behaviour of a simple RNN unit as illustrated in figure 2.2.

$$h_{i+1} = \sigma(W^{hx} x_i + W^{hh} h_i + b_h) \tag{2.2}$$

$$y'_i = \text{softmax}(W^{yh} h_i + b_y) \tag{2.3}$$

We can see in equation 2.2 how the hidden state is carried over. This equation represents the connections across the temporal axis mentioned above. Figure 2.2 shows how the sequence input denoted by x is unfolded and fed into the network one timestep at a time. For each timestep the corresponding output is computed as well as the hidden state that is carried over to the next timestep. Intuitively the hidden state can be seen as the variable

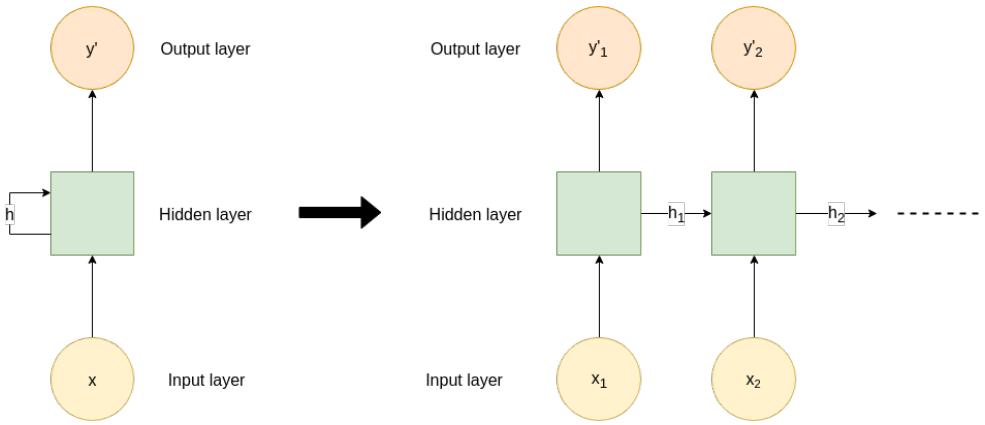

Figure 2.2: Recurrent Neural Network

holding the relevant information from the previous steps to influence the prediction of the next step.

Backpropagation

Backpropagation is the technique used to train neural networks.

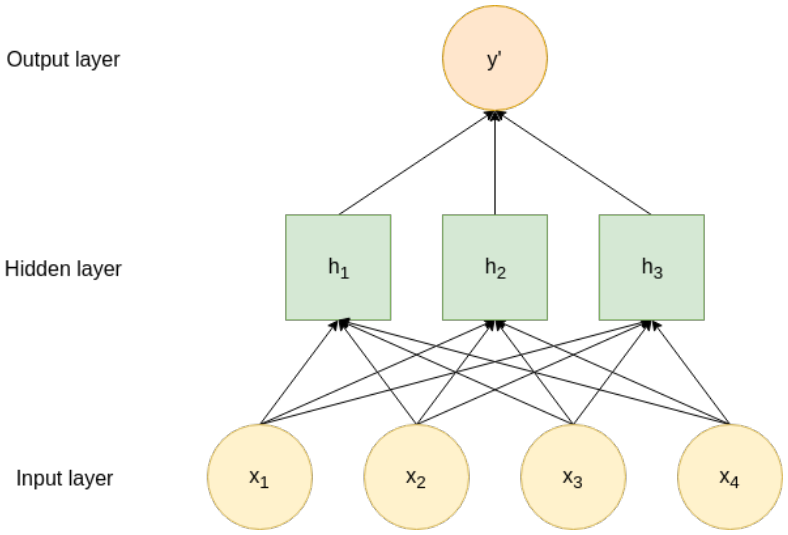

Figure 2.3: Feedforward Neural Network

Let us assume we have a simple feedforward neural network as shown in figure 2.3. This behaviour of this network is described by the following equations.

$$h_i = \sigma_h(w_{h_i}^T x) \tag{2.4}$$

$$y' = \sigma_y(w_y^T h) \tag{2.5}$$

Each individual cell has a weight vector associated with it denoted by w_y in the example of the output unit. The arrows in the illustration represent the individual weights of such a weight vector. Further each layer has an nonlinear activation function associated with it, denoted by σ_l, l being the respective layer.

In this setting we usually are in a supervised mode, where for each input x there is a true label y. Using the neural network we estimate y for a given x by applying the equations above to obtain the estimation y'.

The last component needed before training is possible is a loss function $\mathcal{L}(y, y')$, which quantifies the error of the estimation y'.

The basic idea of backpropagation is to attribute parts of the error computed to the different units of the network, and therefore make adjustements to the weight vector of a particular unit. As described in [4] this is done by using the chain rule to compute the derivative of $\mathcal{L}(y, y')$ with respect to each weight vector. The weight vector is then adjusted by gradient descent, i.e. by moving the vector in the direction of the largest decrease in the gradient. There are more parameters involved such as the learning rate, which defines how much in the direction of the gradient the weights are adjusted, however these are not needed for understanding the principle.

Backpropagation with RNNs

Also in [4] the authors mention that training RNNs has been known to be especially difficult due to the *vanishing* or *exploding gradients* problem. As an example we consider the RNN in figure 2.2. Now we assume we compute the estimation y'_t, i.e. the output of timestep t. As we can see in the equations 2.2 and 2.3 this output is also influenced by all the previous timesteps. The problem now arises from the so called recurrent weights denoted by W^{hh}, those connecting the units across the temporal axis. For now we assume that the weights are small, i.e. $|w_{ij}| < 1$ for $w_{ij} \in W^{hh}$.

Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is a recurrent unit which aims to solve the exploding/vanishing gradients problem mentioned above.

- Explain the concept of recurrent neural Networks
- What are the problems (vanishing gradients etc)
- Explain what a GRU is and why does it handle vanishing gradient better than simple RNN cells?

2.2.2 Embeddings of categorical data

- What is the problem with categorical data?
- What is an embedding, what is the goal?
- How can we compute embeddings?

2.3 Previous Work

2.3.1 Meta-Prod2Vec

- This is a model that allows us to capture the semantic similarity between products
- Explain the assumption of word2vec
- Show equation of word2vec
- How do we apply this to products?

[8]

2.3.2 Hierarchical RNNs for personalized Recommendations

- This is the starting point for the thesis
- Which components does the system have?
- Show the graphic from the paper

[5]

2.4 KPIs

2.4.1 Click-Through-Rate

2.4.2 Conversion Rate

- Describe different KPIs
- What do they meaure, how to optimize for it

Chapter 3

Dataset

3.1 Data Collection

In this work we will use the data generated by the tracking systems of Digitec Galaxus AG. The following sequence-diagram gives an overview of the data collection process.

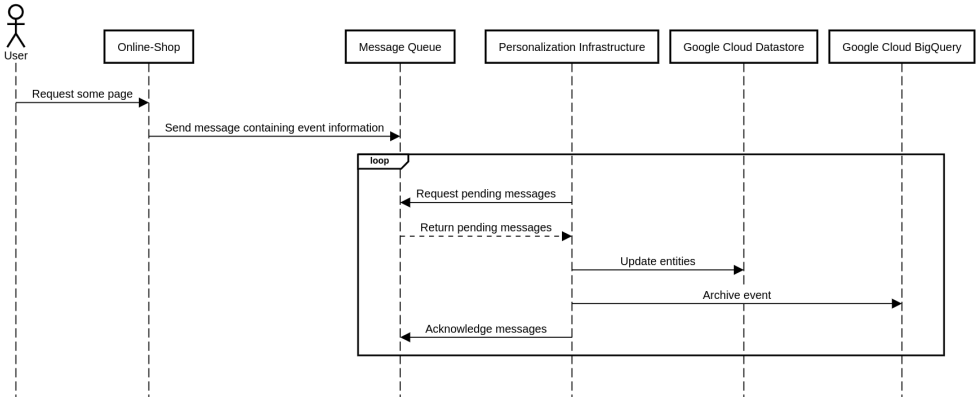

Figure 3.1: Collecting user interaction data on digitec.ch and galaxus.ch

When a user requests a specific page, the Online-Shop Application will collect some information on the user such as UserId, requested page, User Agent etc. This information is packaged as a message representing this specific event and then sent to a message queue. After that the Personalization Infrastructure will process these events by requesting batches of unprocessed messages. For each event then the Personalization Infrastructure will do two things: First it will update the involved entities, second it will archive the event. In the former case a managed Key-Value store

(Google Cloud Datastore) is used to store entities. Examples of such entities are Shopping Carts, Orders, and Last Viewed Products. In principle these entities represent the current state of various entities appearing in the context of the Online-Shop. In the latter case a managed Data Warehousing solution (Google BigQuery) is used, this solution provides the possibility to store large amounts of data in append-only tables. The data stored there is mainly denormalized, such that easy extraction is possible. Each row in these append-only tables contains all information belonging to a single event produced by a user. Essentially the data stored in the Key-Value store is the sum of all events stored in the data warehouse.

3.2 Data Preparation

To be able to focus on the model implementation when implementing the model we want to prepare the data for congestion as far as possible. Therefore the extraction and preparation of the dataset is implemented separately from the model implementation. The process is very similar to the process described in [5].

Data Extraction In the first step we extract the raw data from the data warehouse, selecting the following pieces of information: ProductId, LastLoggedInUserId, UserId, SessionId, User Agent, Timestamp. Most of the properties are self-explanatory, except LastLoggedInUserId. This property represents the UserId last seen on a specific device accessing the Online-Shop, therefore it is useful if the user did not actively log in to the account. In this case we would not know which user accessed the page, however using this property we can complete missing UserIds in the dataset. The data extracted is limited to the events producted by visiting a product detail page as seen in figure 3.2. This filtering is done because this work focuses on recommending products, in another setting other data might also be relevant. The extracted data is stored in several shards of CSV files, each shard approximately represents the events of one day.

Cleaning data In the next step the data is cleaned, which involves mainly two steps. Anytime we encounter an event where we do not know the user, we try to complete the information with the LastLoggedInUserId. If we do not know the LastLoggedInUserId as well we discard the event.
There are many well-known bots roaming the Internet collecting various types of information on websites. Most famously the Google Bot which crawls the content of any website to determine the quality of the site and influence the rank of the website in Google searches. There are some open-source lists that collect the User Agents of these bots. The User Agent of a client accessing a website usually describes the type of device used, in the

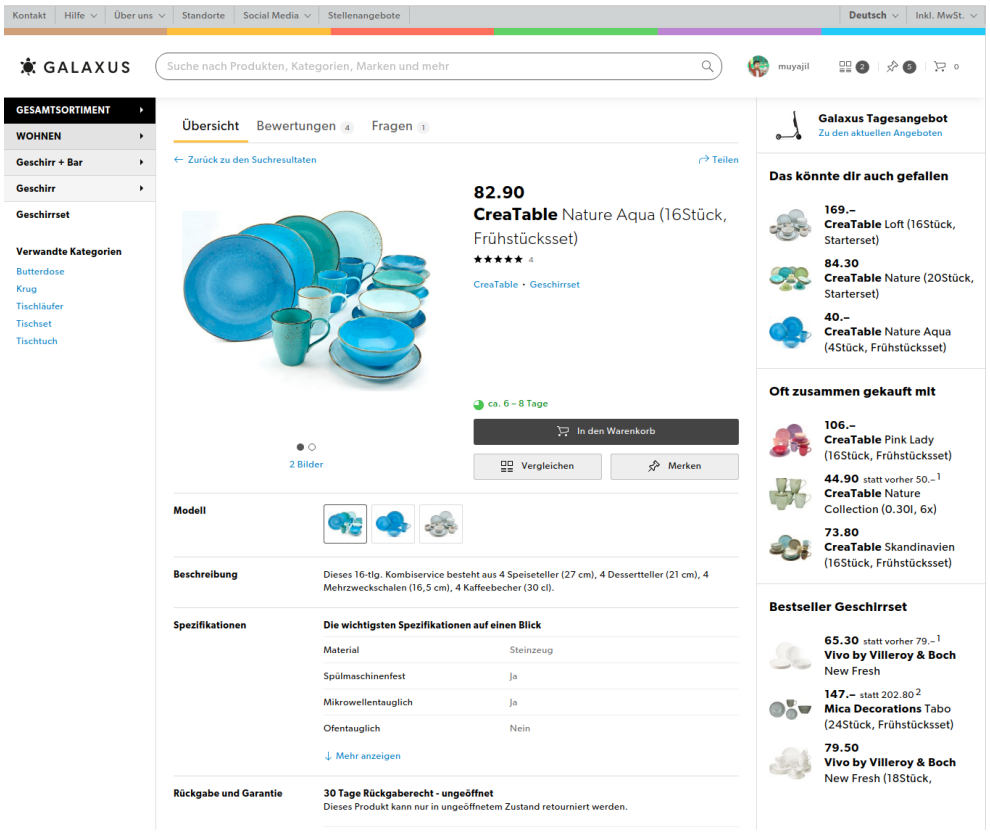

Figure 3.2: Product Detail Page on galaxus.ch

case of ”good” bots they explicitly tell the server that the ”device” accessing the website is actually a bot. Using one of these lists[1] the events produced by bots are removed from the dataset.

Aggregation of events The next step is to aggregate the events on two levels, first grouping events that were generated in the same session and second grouping these sessions by users. The resulting data structure looks as follows:

[1] https://raw.githubusercontent.com/monperrus/crawler-user-agents/master/crawler-user-agents.json

```
    "<UserId_1>": {
        "<SessionId_1>": {
            "StartTime": "<Timestamp of first event>",
            "Events": [
                {
                    "ProductId": "<ProductId_1>",
                    "Timestamp": "<Timestamp_1>"
                },
                {
                    "ProductId": "<ProductId_2>",
                    "Timestamp": "<Timestamp_2>"
                }
            ]
        }
    }

```

Listing 3.1: Data Structure for user-events

This is done by processing one shard after another, when a shard is finished the JSON representation of this shard is saved in a separate file. The reason is that the large amount of datapoints cannot be kept in memory altogether.

Merging shards Since the shards only approximate the events of one day, it is not possible to assume that a session is completely represented in one shard, it might be distributed across multiple. Therefore it is necessary to merge the data in the different shards, however since it is not efficient to keep all the information in one file, a different method of paritioning is needed. However as we will see later it is further necessary that all the events produced by a single user are in the same file. An easy way of achieving this is by partitioning by UserId. Each shard is processed as follows:

```
    for shard in shards:
        for i in range(num_target_files):
            relevant_user_ids = list(filter(lambda x: int(x) %
num_target_files == i, shard.keys()))
            output_path = merged_shards_prefix + str(i) + '.json'
            output_file = json.load(output_path)
            for user_id in relevant_user_ids:
                for session_id in shard[user_id]:
                    # Merge events from output_file and shard

```

Listing 3.2: Merging shards

Filtering Now we have some number of files containing all the information relevant to some users. As the authors of **??** mentioned as well, it makes sense to filter out some datapoints. Specifically the following:

- Remove items with low support (min 5 Events per product)

- Remove users with few sessions (min 5 Sessions per user)
- Remove sessions with few events (min 3 Events per session)

The reasons for removing these is rather obvious. Items with few interactions are not optimal for modeling, sessions that are too short are not really informative, and finally users with few sessions produce few cross-session information.

Subsampling Prototyping models with very large datasets is very inefficient. Therefore several different sizes of the dataset were produced. First the approximate number of users and the exact number of products desired in the dataset is defined. This was done by sampling from the set of products with a probability proportional to the number of events on that product. In a next step the partitions of the data are processed, iterating through the sessions of the users. For each sessions we remove the products that were not chosen to be kept, if the session is still long enough, the session is kept. Further a user is kept in the dataset if the number of filtered sessions is still large enough. This process is repeated for the different partitions until the number of users is larger than the approximate number of desired users.

Embedding Dictionary As we will see in 4.1 a version of the model uses one-hot encodings for representing products. Therefore the product IDs referenced in the dataset need to be mapped into a continuous ID space, otherwise it is not possible to produce one-hot encodings. The process for this is straight forward: Start with EmbeddingId 0 then iterate through all the sessions, and each time a product is seen for the first time it will be assigned the EmbeddingId and the EmbeddingId is increased by 1.

3.3 User Parallel Batches

Since we are dealing with sequence data, it is not trivial to produce batches for the model to ingest. Especially since the sequences can have different lengths, and the number of sessions varies from user to user. User Parallel Batching is a way of having fixed size batches that can be ingested by the model, while allowing for sequences to have different lengths and users to have a different number of sessions. Usually in these cases the sequences are padded with some neutral value, however in this case there is no neutral product, and the introduction of such a neutral product might influence the results. Furthermore when processing sequence data it usually means that a datapoint is the input for the next datapoint, which means we still need to have the ordered sequence data. To illustrate these batches lets assume that there are 4 users with the sessions as in figure 3.3.

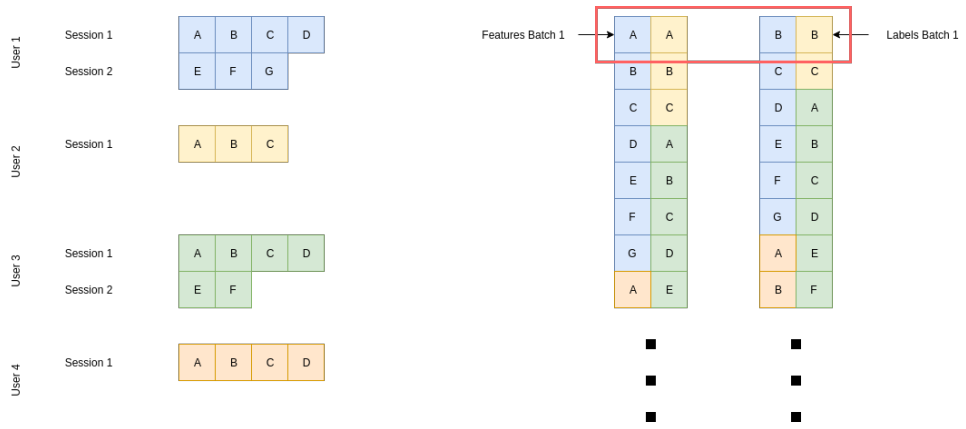

Figure 3.3: User Parallel Batches

In principle a batch of size x will contain an event from x different users. Essentially the labels are the event that happens after the input event.
The data representation shown in listing 3.1 is chosen explicitly to enable efficient generation of user parallel batches.

3.4 Dataset properties

Show dataset stats for different datasets

Chapter 4

System Overview

4.1 Model Architecture

The inspiration from this work comes from [5]. The authors designed a model architecture that can capture

- Describe how it captures the session representation
- illustration for hierarchical neural network
- Describe Model Architecture
- Describe different Components of Model Architecture
- Describe different variants of the model (with/without pf, with/without embeddings)

4.1.1 Model training

4.2 Product Embedding

- Describe the Architecture of MetaProd2Vec
- Write about the features used, what are the transformations etc.

4.3 Implementation

- Describe Class Diagram
- Describe prediction mode/training mode
- Describe problems that arose during training (extensive resources used for so many products and users)

4.3.1 API

As described above the model is implemented in Tensorflow[1]. Tensorflow provides a mechanism to export tensorflow models as so called SavedModel[2], which is the way Tensorflow serializes models universally. They also provide a premade Docker image[3] which allows the model to be served as a REST or GRPC API.

- Describe API

4.4 Production Setup

Digitec Galaxus AG is the largest online-retailer in Switzerland. They operate galaxus.ch and digitec.ch. The former is a general online-shop comparable to Amazon. The latter is specialized in Electronics. Distributed on the different sections of the site there are several recommendation engines populating the content the users see. Examples are the landing pageand multiple engines on the product detail page.

Add screenshot

Add screenshot

There is a framework that computes probabilities which specific recommendation engine provides the content for a specific location. Further the framework then chooses the content for each location based on the probabilities computed before and some other constraints such as minimum and maximum value. However to test the model implemented in this work this framework is bypassed by a A/B Testing engine, therefore this framework is not part of this work. The specific tests and the test setup is described in 5.1.1, for the understanding of the following it is enough to assume that some independant system is providing recommendation requests to the recommendation system. Using the API containers described in 4.3.1 we can serve these requests. The sequence-diagram in figure 4.1 should give an overview on how the system is integrated in the production environment.

[1] http://tensorflow.org

[2] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md

[3] https://hub.docker.com/r/tensorflow/serving

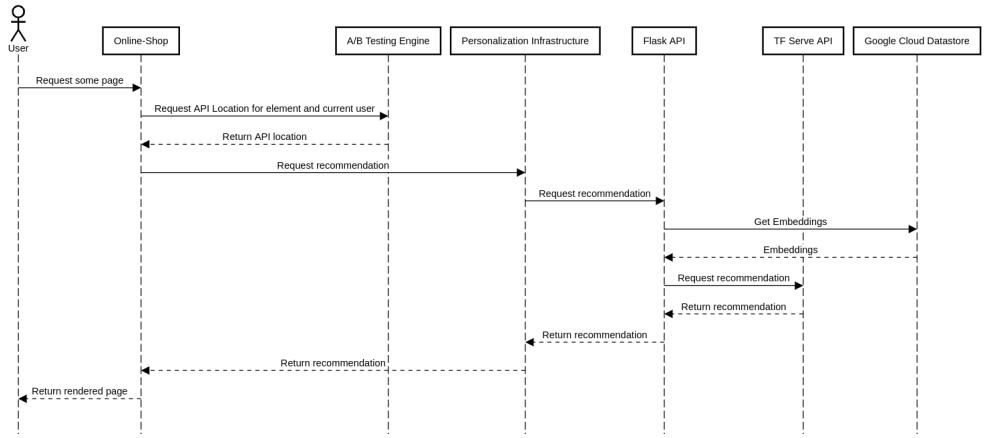

Figure 4.1: Serving Recommendations on digitec.ch and galaxus.ch

The whole process starts when a specific user requests a specific page on either digitec.ch or galaxus.ch. If the requested page has an element where there is an A/B test configured the Online-Shop Application will make a request to the testing engine. As mentioned above the testing engine will then make a request to a API location, which corresponds to one of the model versions. After that the Online Shop will request the content, in this case, from the Personalization Infrastructure. Note that during this setup each of the four versions of the model is deployed simultaneously, all of them trained on the same dataset. The Personalization Infrastructure then calls the API described above. Before requesting a prediction from tf-serve the API will gather precomputed embeddings for the involved entities. The predictions are returned to the Personalization Infrastructure and from there to the Online Shop. The last step is for the Online Shop to render the content and deliver the page to the user. This process takes about XXmsms.

Get ms number here

Chapter 5

Experiments

For each of the model variants evaluate the experiments offline and online.

5.1 Offline

5.1.1 Experiment Setup

- Describe the last session out split (test and eval)
- Describe how we train the model and how we evaluate it

5.1.2 Measurements

- Here we want to confirm a few things: user rnn helps, embeddings helps, profforcing helps
- The evaluation set is always the same
- Show the relevant KPIs (described in the chapter before)

5.2 Online/Production Experiment Setup

5.2.1 Experiment Setup

- Describe the online shop
- Where do we have recommendations, general description
- Explain SOFI
- Where do we put our recommendation system? Against what does it compete?
- Also describe the testing engine

5.2.2 Measurements

- First measure for each variant what the performance is (an ABCD test so to say, split customers maybe, there should be enough)
- Then explain the performance of Simple models
- After that the performance of a black box, specifically Google's AutoML
- Show results for AutoML, Often Bought Together and our system in the different variants
- Show the relevant KPIs

Chapter 6

Results

6.1 User Embeddings

6.2 Session Embeddings

6.3 Product Embeddings

- bring everything together
- compare best performing model to automl from google

Bibliography

[1] Christopher R. Aberger. Recommender : An analysis of collaborative filtering techniques. http://cs229.stanford.edu/proj2014/Christopher%20Aberger,%20Recommender.pdf, 2014.

[2] Erik Bernhardsson. Music discovery at Spotify. https://www.slideshare.net/erikbern/music-recommendations-mlconf-2014, 2014.

[3] Paul Covington, Jay Adams, and Emre Sargin. Deep Neural Networks for YouTube Recommendations. https://ai.google/research/pubs/pub45530, 2016.

[4] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. https://arxiv.org/abs/1506.00019, 2015.

[5] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. http://arxiv.org/abs/1706.04148, 2017.

[6] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to Recommender Systems Handbook. http://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf, 2011.

[7] Marcel Urech. Migros CIO Interview. https://www.netzwoche.ch/news/2017-06-29/martin-haas-ueber-cumulus-twint-und-die-migros-it, 2017.

[8] Flavian Vasile, Elena Smirnova, and Alexis Conneau. Meta-Prod2Vec - Product Embeddings Using Side-Information for Recommendation. http://arxiv.org/abs/1607.07326, 2016.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.