HAW HAMBURG

MASTERTHESIS
Timo Lange

Deep Learning Based News Recommendation Using Text and Metadata

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Timo Lange

Deep Learning Based News Recommendation Using Text and Metadata

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: July 12, 2022

Timo Lange

Thema der Arbeit

Deep Learning Based News Recommendation Using Text and Metadata

Stichworte

Deep Learning, Maschinelles Lernen, Empfehlungen, Nachrichten, NLP

Kurzzusammenfassung

Empfehlungssysteme (RS) sind weit verbreitet und werden in vielen Bereichen eingesetzt, z.B. bei der Empfehlung von Artikeln im elektronischen Handel, beim Musik- und Videostreaming oder in Nachrichtenportalen. In dieser Arbeit wird ein proprietärer Datensatz für Nachrichtenempfehlungen vorgestellt, für den eine Basis an Messergebnissen erstellt und verschiedene Fragen behandelt werden, z.B. ob die Schlagzeile, der Teasertext oder der vollständige Artikeltext für Nachrichtenempfehlungen geeignet sind. Ein immer größerer Teil der *RS*-Forschung befasst sich mit Deep Learning (DL), daher wird sich diese Arbeit auf diesen Bereich von *RS* konzentrieren. Um ein geeignetes Empfehlungsmodell auszuwählen und festzustellen, welche Methoden zur Darstellung des Artikelinhalts verwendet werden können, wird eine breite Auswahl an aktuellen *DL*-basierten Sprachmodellen (LM) und *RS* betrachtet. Im Zuge der Implementierung des Modells und der Verarbeitungspipeline werden die Schwierigkeiten im Umgang mit einem großen realen Nachrichtendatensatz untersucht. Es stellte sich heraus, dass für das gewählte Modell der Teaser-Text am besten für Nachrichtenempfehlungen geeignet ist. Es hat sich ebenfalls gezeigt, dass bei der Verarbeitung eines realen Datensatzes die große Datenmenge und die hohe Dimensionalität der Daten die größte Herausforderung darstellten. Schlussendlich hat die Analyse der Daten das große Potenzial für personalisierte Nachrichtenempfehlungen aufgezeigt, da die meisten Artikel nur wenige Seitenaufrufe haben und viele Nutzer nur ein bis zwei Artikel ansehen. Daher könnten sehr gezielte Empfehlungen die Anzahl der Seitenaufrufe vieler Artikel, die eine sehr kleine Zielgruppe haben, erheblich steigern.

Timo Lange

Title of Thesis

Deep Learning Based News Recommendation Using Text and Metadata

Keywords

Deep Learning, Machine Learning, Recommendation, News, NLP

Abstract

Recommendation systems (RS) are widely used and prevalent in many areas like recommending items in e-commerce, music and video streaming, as well as news portals. In this thesis, a proprietary news recommendation dataset is introduced for which a baseline will be established and several questions will be approached, like whether the headline, teaser text or full article text is suitable for news recommendation. An ever growing amount of *RS* research is going in the direction of deep learning (DL), so this work will focus on this domain of *RS*. To select an appropriate recommendation model and determine which methods can be used to represent the article content, a broad selection of current *DL* based language models (LM) and *RS* are reviewed. In the course of implementing the model and the processing pipeline, the difficulties in handling a large real world news dataset are examined. It turns out, that for the selected model, the teaser text worked best for news recommendation. Also it has shown, that in handling a real world dataset, the large amount of data and the high dimensionality of the data posed the biggest challenge. Lastly, the analysis of the data unveiled the great potential for personalized news recommendation, as most articles have just a few pageviews and many users just view one to two articles. So very targeted recommendations could significantly rise the pageview count of many articles, which have a very niche target audience.

Contents

List of Figures

List of Tables

1 Introduction

Mankind is in the age of information where everybody is confronted with an ever growing amount of information. In the era of *big data* and the overwhelming amount of information users are confronted with, Recommender Systems (RS) can help to overcome the *information overload*. Ricci et al. describe "Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user."[42] This can be supplemented by Jannach et al. with "The construction of systems that support users in their (online) decision making is the main goal of the field of recommender systems. In particular, the goal of recommender systems is to provide easily accessible, high-quality recommendations for a large user community."[21] Also [41] add their description of RS with the distinction of different roles RS play on behalf of the service provider and the user. The users expects relevant item suggestions which meets their personal taste and needs whereas the service providers expectations from the RS are in the direction of increase of sold items, higher turnover, sell more diverse items, increase user satisfaction and fidelity. RS are mainly classified into Collaborative Filtering (CF) based, Content-Based, Knowledge-Based and Hybrid Systems. This work uses deep learning for Collaborative Filtering and Content-Based recommendations combined to a Hybrid System. Whereby systems using CF to provide suggestions to a user rely on the assumption that users with similar interests in the past share their interest also in the future. If two users share a very similar history of liked items, if one user likes a new item the other user may also be interested in this item. Whereas Contend-Based recommendations rely on context data for the items, like item descriptions for products or the topic and content of a news article.

The primary goal of this thesis is to examine news recommendation methods and techniques in the context of a proprietary news recommendation corpus. The corpus is composed of news articles and pageviews. The news articles of the dataset are in the German language and contain the full content body and additional metadata like publication date and publisher. A pageview represents an article view of a user with additional

metadata like start time to read, engagement time, article publisher, and portal where the article was published. A comprehensive analysis of the dataset is carried out in section 3.1. Publicly available datasets usually do not contain the full content body and have a relatively small amount of articles. Additionally, most datasets contain articles in the English language. Also, most public available news datasets have viewer metadata about the articles and pageviews. Furthermore, the size of the dataset is remarkable with 313 565 551 pageviews from 56 199 311 users and 823 947 corresponding articles. The recently published MIcrosoft News Dataset (MIND) [62] for news recommendation research is a big step into the direction of a publicly available big real world news dataset but is nevertheless not comparable in size and extend to the here examined dataset. The examination of news recommendation methods is addressed by extensively analyzing current recommendation techniques and especially news centered recommendation algorithms. Non news specific methods are investigated for their adaption ability to the field of news recommendation. Furthermore, methods for representing news content are explored, as modeling of news is a key component for news recommendation systems. To this extend, an extensive amount of current Language Models are analyzed.

1.1 Research Question & Contribution

The questions this master thesis tries to answer are:

1. **Q1**: How to establish a baseline with the German news recommendation dataset using deep learning?
2. **Q2**: How is the dataset structured and how to take advantage of specific properties of the dataset for recommendation?
3. **Q3**: What are the challenges in utilizing a real world dataset for a RS for design and implementation in contrast to academic datasets?
4. **Q4**: What are the challenges in the adaptation of a general RS model to the news recommendation domain?
5. **Q5**: Which techniques can be utilized to incorporate metadata into the news RS?
6. **Q6**: Does the use of metadata yield better recommendations?

7. **Q7**: Does the full article content yield significantly better recommendation performance compared to headline/teaser text as article content?

The main contributions of this thesis are as follows:

1. Introducing a large real world news recommendation dataset with German article texts.
2. Adapting a recommendation model of the e-commerce domain to the news recommendation domain.
3. Unveil challenges in the use of real world data.

1.2 Outline

This thesis is split into five chapters. **Chapter one** gives a general introduction to this work. In **chapter two** *NLP* methods, in particular *Language Models*, are introduced which can be leveraged for recommender systems, especially for news modeling. The second chapter also gives an introduction to recommender systems with a special focus on *Deep Learning* based systems and news recommendation, where notably works are introduced. **Chapter three** presents the experimental design for the investigations of this work. In the **forth chapter** the evaluation of the experiments is done. The **last chapter** concludes this thesis and gives an outlook to future directions.

2 Analysis

Problem Statement Recommender Systems are a broad field which spans a wide collection of different methods. To this extend, Recommender Systems in general are introduced which is complemented with an additional specialization on news recommendations. Since this work focuses on Deep Learning based recommendations, this chapter presents Deep Learning algorithms and methods used by RS systems. Before this chapter is discussed at the end, a brief review of related work is conducted. Starting with RS related work, a small section is dedicated to Natural Language Processing (NLP). As this work deals with text data, NLP methods are inevitable to access the information contained in the text.

2.1 NLP

Natural Language Processing (NLP) is closely connected with recommender systems. Either as a tool for content extraction from texts or as adaptations of sequence processing algorithms developed for NLP to process sequences in recommendation tasks. Figure 2.1 from [47] shows a timeline with some notable steps, how NLP techniques have influenced the field of RS.

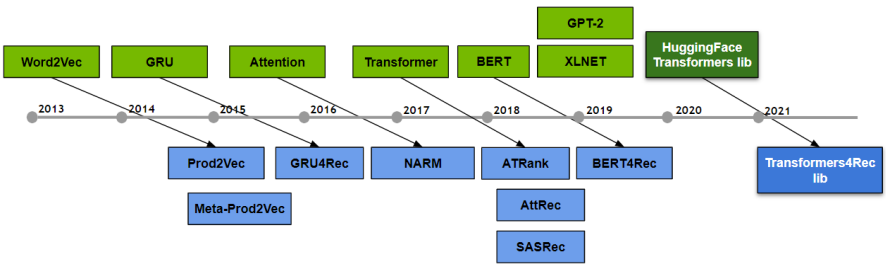


Figure 2.1: "A timeline illustrating the influence of NLP research in Recommender Systems" [47]

An overview of current *DL* based *NLP* methods is given by [52].

To leverage the information of news articles (or text documents) for recommenders, capture their characteristics, and understand their content, it is crucial to model these articles. News modeling techniques can be roughly divided into two categories, which are *feature-based* and *deep learning-based*. *Feature-based* techniques are not learned from scratch and mainly represented by handcrafted features, which is time consuming and costly. To this extend, this work is focused on deep learning methods to automatically learn news representations. These deep learning techniques comprise mostly of neural NLP methods to learn news representations from news texts. [58] Language models are a field of *NLP*. So in this section, the utilized NLP methods are sketched out as well as some notable Language Models, which are used by RS described in section 2.2.1. In [5] is shown, that language models aggregate much knowledge when trained on a large dataset and equipped with many parameters, so specific tasks can be solved in a few shot scenario. Furthermore, Petroni et al.[34] investigate in how far pre-trained LM can be seen as knowledge bases and find that *BERT* contains relational data, works well on open-domain question answering, and learns certain types of factual knowledge. This indicates that deep learning based language models are able to extract rich information from news text to leverage for recommendations and may be a crucial building block for news modeling and recommendation.

2.1.1 Language Models

"Language modeling is the art of determining the probability of a sequence of words." [14] More precisely "Language modeling is the task of assigning a probability to sentences in a language [...]. Besides assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words [...]." [65, p. 105] Furthermore, "The notion of a language model is inherently probabilistic. A language model is a function that puts a probability measure over strings drawn from some vocabulary." [31, p. 238] So *Language Models (LM)* are a probabilistic method to learn from a text corpus how a language (or multiple languages) are constructed. LM in combination with *Transfer Learning* can be a powerful technique for downstream tasks as shown by [19]: "Language modeling can be seen as the ideal source task and a counterpart of ImageNet for NLP: It captures many facets of language relevant for downstream tasks, such as long-term dependencies, hierarchical relations, and sentiment. In contrast to tasks like MT and entailment, it

provides data in near-unlimited quantities for most domains and languages. Additionally, a pre-trained LM can be easily adapted to the idiosyncrasies of a target task, which [...] significantly improves performance [...]. Moreover, language modeling already is a key component of existing tasks such as MT and dialogue modeling. Formally, language modeling induces a hypothesis space H that should be useful for many other NLP tasks." [19] Furthermore, "LMs have played a key role in traditional NLP tasks such as speech recognition, machine translation, or text summarization. Often (although not always), training better language models improves the underlying metrics of the downstream task (such as word error rate for speech recognition, or BLEU score for translation), which makes the task of training better LMs valuable by itself." [23]. For this reason, it is important to understand how current LM work and how they can be incorporated into RS.

Next, some notable recent LM are presented. This list is by far not comprehensible as so many LM were proposed recently, e.g. LM which have remarkable results on benchmarks but have a huge amount of parameters and are essentially scaled up versions of previous models, without substantially improved methods, are left out. This applies also for LM whose improvements in benchmarks are essentially due to training on bigger datasets. As the LM must be applicable to the RS domain, only models which work with a reasonable amount of parameters are considered, due to the real time requirements at inference time, like low latency and high throughput of recommendations. Methods to scale LM to large sizes and train them efficiently despite large parameter sizes are shown by [37]. Techniques to explicitly make transformer based models more efficient and independent from scaling, is examined by [46]. The letter used evolutionary search to find efficient architectures in the search space of combinations of *Tensorflow (TF)* primitives. Their findings can be dropped into existing code bases without further modifications and may be useful for *transformer* based RS, either to reduce hardware requirements or make more complex models possible. The LM also have to be integrated into the RS, which can be done via *embeddings* of the texts. An example of how sentence embeddings can be extracted from *BERT* is given by [39].

ELMo

The *ELMo (Embeddings from Language Models)* model by [33] is the first LM which uses the whole context of a word (all preceding and following words) as context to compute word embeddings, so that it is a function of the entire input sentence. ELMo learns

high quality representations of words and addresses the challenges of "(1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy)" [33]. ELMos vectors are learned functions of the internal states of a deep bidirectional language model (biLM) based on LSTMs, which is pre-trained on a large text corpus with a coupled LM objective. Here a biLM combines a forward and backward LM, so it computes the input sentence from the beginning to the end and from end to beginning. Furthermore, with the formulation of [33] the log likelihood in the forward and backward directions is jointly maximized. The learned representations can be added to existing models to enhance their performance over the usage of e.g. simple word embeddings. The representations are a function of all internal layers of the biLM, which differentiates the method from previous ones, which uses only the top LSTM layer. For each end task, a linear combination of the vectors stacked above each input word (intermediate layer representations in the biLM) is learned.

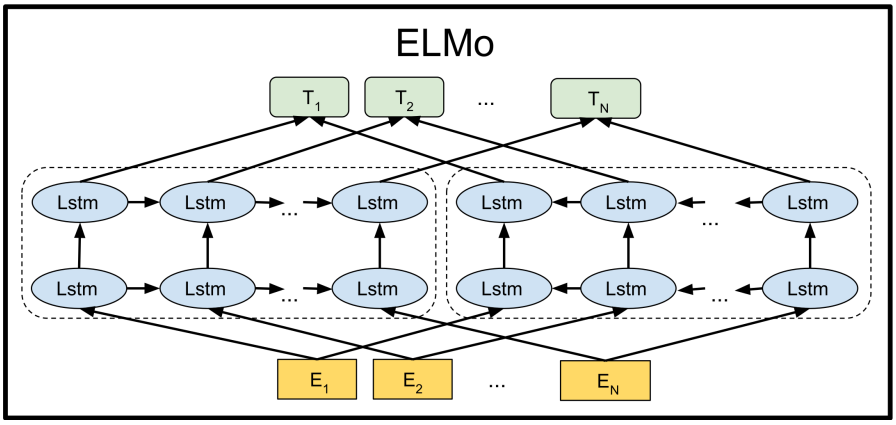


Figure 2.2: ELMo pre-training model architecture: "ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks" [11]

The architecture of ELMo is shown in figure 2.2. The parameters for both the token representation and Softmax layer in the forward and backward direction are tied while the parameters of the forward and backward LSTMs are separated. Furthermore, residual connections between the LSTM layers are added. For each token of the input, the L-layer biLM computes a set of 2L + 1 representations. For usage in a downstream model, all layers of the representations are collapsed into a single vector.

BERT

Bidirectional Encoder Representations from Transformers (BERT) is a LM which is pre-trained with the *Masked Language Modeling (MLM)* objective and can be finetuned with one additional output layer for a wide range of downstream tasks without profound architecture changes specific for the task. The *MLM* objective is similar to the Cloze task, where the blanks in a sentence have to be filled, e.g. "Mice likes to eat _". It is designed for unsupervised pre-training of text for deep bidirectional representations, which are jointly conditioned on left and right context. At its introduction, *BERT* achieves state-of-the-art (SOTA) results in many *NLP* tasks and is the basis for many later announced architectures which further pushed the SOTA on many *NLP* problems. *BERT* also got adapted to the RS domain by [48, 3, 66], see section 2.2.1 for more details.

Figure 2.3 shows the overall architecture of *BERT*. From bottom to the top, the input tokens get embedded first, then several transformer layers follow, and at the end the output tokens are generated for every input token. The *Transformer* in the architecture is a multi-layer bidirectional *Transformer* encoder and is based on the original *Transformer* [53].

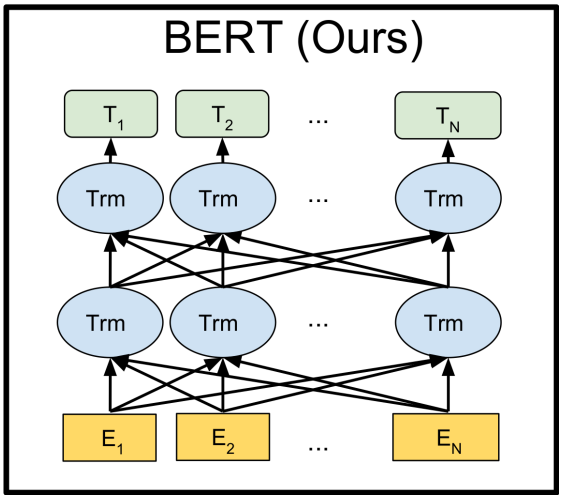


Figure 2.3: BERT pre-training model architecture: “BERT uses a bidirectional Transformer. [...] BERT representations are jointly conditioned on both left and right context in all layers” [11]

Figure 2.4 shows the pre-training and fine tuning architecture. At pre-training *BERT* is fed with a sequence of two sentences and two special tokens. The *[CLS]* is a special classi-

fication token whose final hidden state represents the aggregated sequence representation and is used for classification tasks. The *[SEP]* symbol is a special separation token which separates the first sentence A and second sentence B. At fine-tuning time, the *[SEP]* can be either used as separator for the specific task, e.g. to separate the question from the context in a question answering task, or can be left out for tasks which require just a sequence of a single sentence.

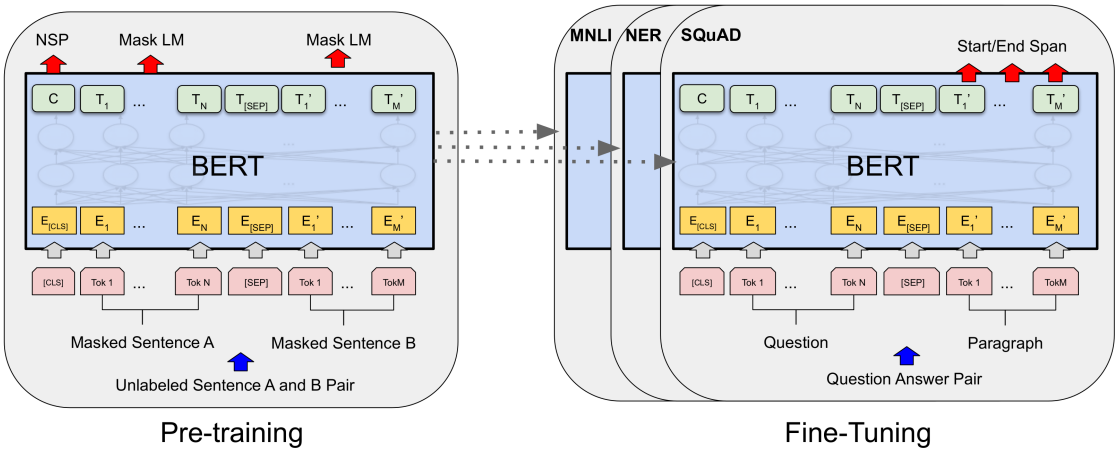


Figure 2.4: BERT pre-training and fine tuning [11]

Figure 2.5 shows the aforementioned input embeddings, which uses *WordPiece* embeddings, where words are further separated into pieces like "play" and "ing" for "playing", which reduces the vocabulary size substantially and preserves more information in contrast to stemming. The input embedding constitutes of three different embeddings, which are concatenated for each token. The embedding of the token itself, an embedding which indicates to which sentence it belongs to and a positional embedding. The positional embedding is needed as *Transformers* have no intrinsic sense for sequences like recurrent models, so the order of the sequence have to be encoded.

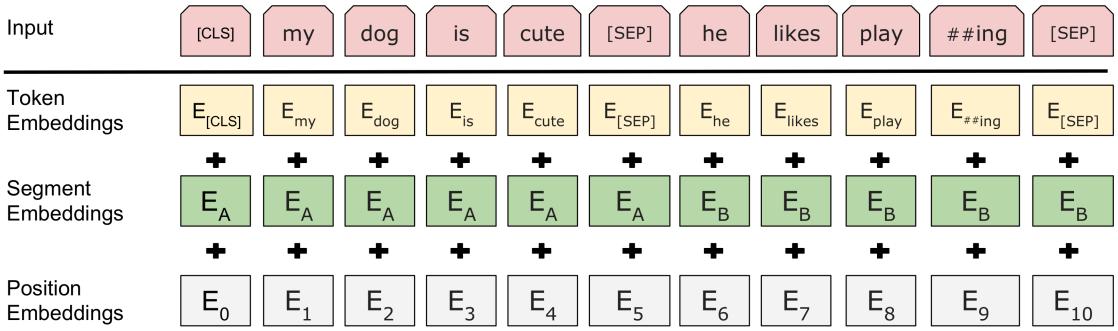


Figure 2.5: BERT input representation [11]

BERT has two training objectives: Firstly, the MLM and secondly a *Next Sentence Prediction (NSP)* objective. The MLM task is used to train on the left and right context of the masked token, which separates *BERT* from *autoregressive (AR)* models like GPT. 15 % of the tokens are replaced with *[MASK]* by 80 % chance, 10 % by a random token and 10 % left unchanged. The random replacement/unchanged strategy is used as the *[MASK]* token is not used during fine-tuning time. This creates a mismatch between pre-training and fine-tuning and the strategy mitigates this. The *NSP* objective serves the model to learn the relationship between two sentences, which is not directly addressed by the LM objective. The *NSP* is simply achieved by choosing the sentence B 50 % of the time to be the next sentence and otherwise chosen to be a random sentence and the model has to predict whether it is the next sentence or not.

BERT Family In this paragraph, some BERT derivatives/improvements are briefly listed:

SpanBERT *SpanBERT* is a pre-training method, which advances the training of the *BERT* model to better predict and represent spans of text. It masks random spans of text instead of single tokens. Further, it trains to predict the span boundaries representations to predict the whole content of the masked span, where only the boundary tokens of the span are used. Also the NSP objective is omitted and a single contiguous sequence is used per sample. These changes in pre-training improves the basis *BERT* model for many tasks. [22]

StructBERT *StructBERT* incorporates language structures into the pre-training with two auxillary tasks, which leverages word and sentence structures. At first, after word

masking, a certain number of tokens are shuffled randomly and the model has to predict the right order. Secondly, sentences are randomly swapped and the model has to predict which is the next and previous sentence. The two new objectives are supposed to better encode the dependencies between words and sentences and rise the ability for generalization and adaptation of the model. [56]

RoBERTa *Robustly optimized BERT approach (RoBERTa)* introduces an improved pre-training for *BERT* models. Liu et al. found that *BERT* was significantly undertrained and improved results with longer training with bigger batch sizes on more date. To increase the amount of training data they collected a large news dataset called *CC-NEWS* Furthermore, the NSP objective was removed and the model is trained on longer sequences. Additionally the masking pattern on the training data is changed dynamically, where *BERT* generated masks initially once for the hole dataset and used the same masking per sentence over the whole pre-training. This showed, the MLM objective alone is competitive with other proposed pre-training methods. Also the Robustly optimized BERT approach following [36] is used. [30]

ALBERT *A Lite BERT (ALBERT)* introduces two parameter reduction techniques to reduce the memory consumption and speed up the training of *BERT*, which leads to SOTA results with fewer parameters as the *BERT-large* model. The first technique is to factorize the large vocabulary embedding matrix into two smaller matrices. Secondly, cross-layer parameter sharing is utilized, thus deeper networks do not rise the parameter count. A *BERT-large* like model has 18 times fewer parameters and trains 1,7 times faster. These techniques also function as regularization and stabilize the training as well as improving generalization. Additionally, the *sentence-order prediction (SOP)* objective is utilized which focuses on inter-sentence coherence to overcome the ineffectiveness of the *NSP* objective from *BERT*. [27]

DistilBERT *DistilBERT* uses knowledge distillation during pre-training which reduces the model size by 40 % and retains 97 % of the language understanding capabilities, which comes also with a speedup of 60 %. The model has half the layers of *BERT-base* and the same size of hidden units per layer. The layers of the distilled model are initialized with every second layer of the original model. The distilled model is trained with three linear combined losses. a) The distillation loss over the soft target probabilities the teacher

(original model), b) the MLM loss and c) a cosine embedding loss, which aligns the hidden state vector of the original and distilled model. [44]

DeBERTa *Decoding-enhanced BERT with disentangled attention (DeBERTa)* surpasses for the first time the human performance on the SuperGLUE benchmark. It uses two novel techniques: The *disentangled attention* mechanism and an *enhanced mask decoder*. The *disentangled attention* separates the embeddings for content and position into two vectors, and the attention is computed using the content and relative position embedding. The final attention score of two words is the sum of four attention scores between *content-to-content*, *content-to-position*, *position-to-content*, and *position-to-position*. The *enhanced mask decoder* supplies absolute position information after all transformer layers and before the softmax layers, which helps the model to distinguish masked words with the same relative context, e.g. "a new **store** opened beside the new **mall**" where **store** and **mall** masked. Additionally, a new virtual adversarial training method called *Scale-invariant-Fine-Tuning (SiFT)* is used for fine-tuning. It is a regularization method to enhance the models generalization through more robustness to adversarial examples by perturbations to the normalized input word embedding. [15]

GPT

The *Generative Pre-Training (GPT)* model is a unsupervised pre-training LM with task specific supervised fine tuning. The architecture of *GPT* is shown in figure 2.1.1. From bottom to the top, the tokenized input words are embedded, which are further processed by several *Transformer* decoder layers, which finally produce the output tokens. As can be seen from the arrows, tokens can only attend to the tokens to the left and did not see the tokens to the right. This makes an autoregressive LM possible. The model is trained on contiguous sequences of text without any additional tokens to learn long range dependencies. For the input, BPE is utilized, the tokenization is done via *SpaCy*, and data is cleaned with the *ftfy* library. For fine-tuning, an additional linear layer followed by a softmax-layer is added, which predicts the labels of the labeled task specific dataset. Additionally, an auxiliary objective during fine-tuning is added to improve generalization and accelerating the convergence. Beside the extra linear layer, only embeddings for delimiters add extra parameter to the task specific model. For fine-tuning on structured inputs, the data is converted into an ordered sequence, seperated by special tokens like *[Start]*, *[Delim]* and *[Extract]*. Radford et al. conclude, that the pre-training compared

to only training on the specific task improves the performance substantially and trough the pre-training on a diverse corpus with long stretches of contiguous text the model acquires a significant world knowledge and the ability to solve long-range dependency tasks which is transferred to downstream tasks. [35]

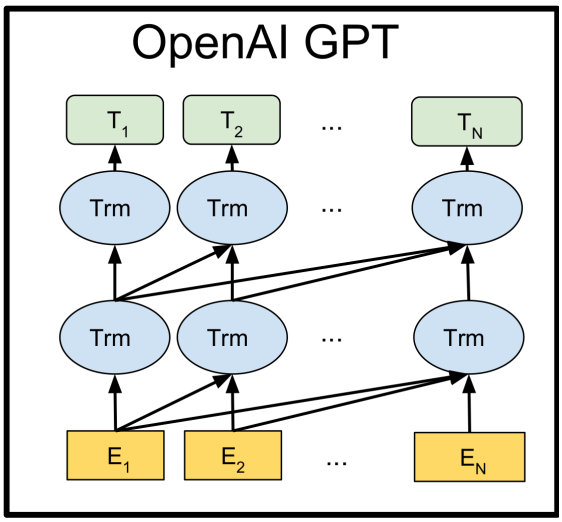


Figure 2.6: GPT pre-training model architecture: "OpenAI GPT uses a left-to-right Transformer" [11]

The *GPT* model received two updates in further work which are briefly described below.

GPT-2 The main difference to the first version lies in a bigger, more diverse dataset, which is crawled from outbound *Reddit* links above a rating threshold to ensure a certain quality of the text, and an order of magnitude more parameters as the first version to 1,5 billion. Architectural changes include the move of *layer normalization* into the input of each sub-block, additional *layer normalization* after the final self-attention block, and adaptation of the initialization accounting for the changes. The aim of the paper is to show how big LM can learn to perform specific tasks without fine-tuning when they are big enough and trained on a large corpus. [36]

GPT-3 The major difference of the model is again a rise in parameter count to 175 billion, and the only change in architecture is the use of alternating dense and locally banded sparse attention patterns in the *Transformers* layers. The authors demonstrate that a big

enough LM can perform well in a *few-shot* and *zero-shot* setting without any additional supervised training on labeled data. [5]

XLNet

To account for *BERT* did not use the dependencies between masked tokens and the pretrain-finetune discrepancy, *XLNet* is a generalized AR pre-training method, which addresses the pretrain-finetune discrepancy with its AR formulation, combined with learning bidirectional contexts by train over all permutations of the factorization order of the tokens with maximization of the expected likelihood. Architecture wise *XLNet* integrates the segment recurrence and relative encoding of *TransformerXL*. [10] To be AR and also have a bidirectional context, the *Permutation Language Modeling Objective* and *Two-Stream Self-Attention* are crucial for the model.

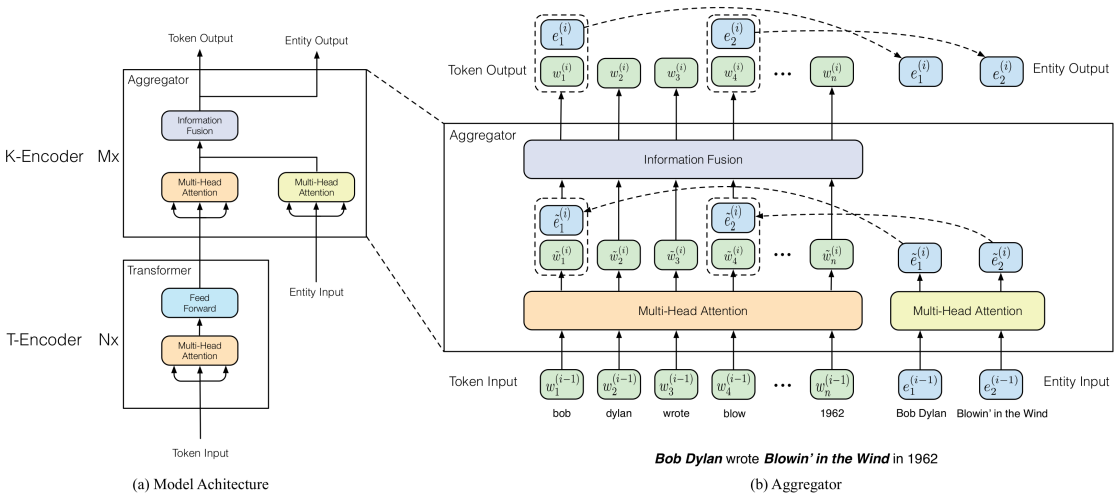
Permutation Language Modeling Objective To be AR and also have a bidirectional context, the factorization order of a sequence is permuted but not the ordering of the sequence. Meaning the position of each token remains the same but the context to predict a token is permuted. E.g. the sequence [1,2,3,4] is permuted to [3,2,1,4] where the number represents their position, then token 3 has no context, token 2 has context [3], token 1 has context [3,2] and token 4 has context [3,2,1]. This is achieved through a relative position encoding for each token and a proper attention mask accounting for the right context for each token.

Two-Stream Self-Attention To be able to compute the permutation LM, each token has a separated embedding for the content and the position of the token. To predict a token, the attention has only access to the position embedding of that token and not its content embedding, which would make the prediction trivial. Also without access to the position encoding, the model would reduce to a simple bag-of-words model. The attention is also masked to only see tokens before in the current permutation order. [64]

ERNIE 1/2/3/TITAN

***E**nhanced Language **R**epresentatio**N** with **I**nformative **E**ntities (ERNIE)* uses a mostly standard *BERT* model combined with a knowledge injection module, which incorporates embedded entities from a *Knowledge Graph (KG)* into the LM. This improves the performance on knowledge-driven tasks while maintaining the score of *BERT* on other common *NLP* tasks.

Figure 2.7 shows the architecture of *ERNIE*. On the left bottom, the input tokens are processed by the textual encoder (*T-Encoder*), which comprises of multiple transformer layers pretty much the same as the *BERT* model. The output of the *T-Encoder* is further processed by the knowledgeable encoder (*K-Encoder*), which comprises of several stacked *Aggregator* layers. The *Aggregator* has two multi-head self-attentions, one for the input token embeddings and one for the entity embeddings. The output of both attention layers, with aligned input token and entity embeddings, is fed into the *Information Fusion* layer, which integrates the heterogenous information and outputs the input token and entity embeddings, which now have both information mutually integrated. The outputs of the top *Aggregator*, both input token and entity embeddings, are then used as features for specific tasks. The *Information Fusion* layer basically consists of two dense layers for the input tokens and the entities, which share one hidden layer, where the token and entity embeddings, after multiplication with the wight matrix, are added up and have a common bias vector.



(a) Model Achitecture (b) Aggregator

Figure 2.7: ERNIE model architecture with textual encoder (*T-Encoder*) *Transformer* and knowledgeable encoder (*K-Encoder*) on the left and one *Aggregator* layer of the *K-Encoder* on the right. [71]

To generate the entities, the knowledge embedding model TransE [4] is trained on Wikidata[1]. These entities are then aligned with the entities in English Wikipedia, which are annotated via TAGME [13]. Then English Wikipedia with the *KG* from Wikidata is used for pre-training, where the underlying Transformer blocks are initialized with the weights of the original *BERT* model released by Google[2].

For pre-training the three objectives NSP, MLM, and denoising entity auto-encoder (dEA) are used. The *dEA* objective is introduced by *ERNIE* to inject knowledge from informative entities into the language representation by randomly masking some token-entity alignments and let the model predict the aligned entities from the corresponding tokens. [71]

ERNIE 2.0 In *ERNIE 2.0*, a continual pre-training framework is introduced to better capture, beside the common co-occurrence of words or sentences, lexical, syntactic and semantic information from the training corpora like named entities, semantic closeness, and discourse relation. To accomplish this, the authors incrementally build pre-training tasks and train pre-trained models on it via continual multi-task learning. This enables to surpass the performance of *BERT* and *XLNet* on several NLP tasks.

[1] https://www.wikidata.org/
[2] https://github.com/google-research/bert

Figure 2.8 shows the differences in *Sequential (Continual) Multi-task*, *Multitask* and *Continual Learning*. In *Continual Multi-task Learning* the model is trained on one task for some iterations. Afterwards the model is initialized with the trained weights and is trained on another task but keeping the previous task in training. This procedure is repeated for every new task and all preceding tasks are jointly trained again with the new task. This is to not forgetting the knowledge gained on the previous task, how it may happen by just *Continual Learning*, and the joint training on multiple task aid one another for better learning from the data.

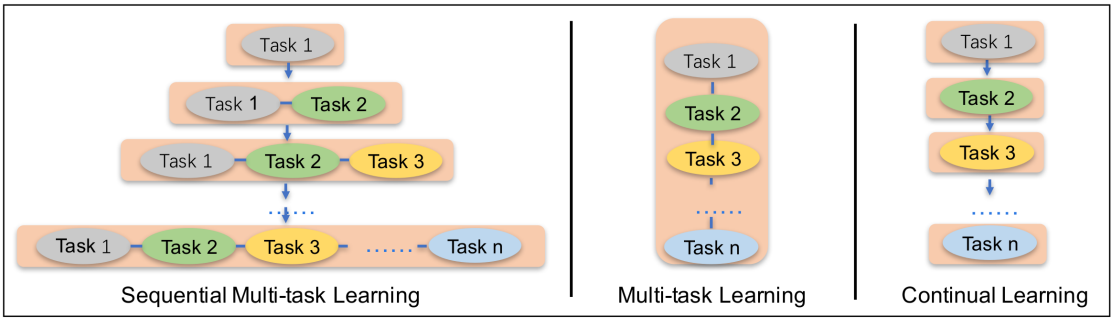


Figure 2.8: Three different learning methods on multiple tasks. **Left**: Sequential (Continual) Multi-task Learning, where after learning a task, a new one is added and the model is trained on the previous and new test, continual adding new tasks. **Middle**: Multitask-Learning, where all tasks are learned jointly at once. **Right**: Continual Learning, where all tasks are learned sequentially. [50]

ERNIE 2.0 is trained on three different kind of tasks to grasp diverse aspects of information. The word-aware tasks, structure-aware tasks, and semantic-aware tasks to capture lexical, syntactic, and semantic information. [50]

ERNIE 3.0 *ERNIE 3.0* uses an architecture, where a universal representation is learned by a *Universal Representation Module*, which is basically a network like *ERNIE 2.0* with the same pre-traing task, except the standard *Transfromer* is exchanged for a *TransformerXL*. [10] The novelty is the usage of *Task-specific Representation Modules* for fine tuning. There are two modules for fine tuning, one *natural language understanding* and one *natural language generation* module. These modules are also *TransformerXL* based but with fewer parameters then the *Universal Representation Module*. During fine-tuning, only the parameters of the *Task-specific Representation Modules* are up-

dated. Also the parameter count for the model is rised to 10 billion and the amount of training data is increased to 4 TB. [49]

T5

The authors seek to explore current techniques in *NLP* like architecture, pre-training objectives, and transfer approaches. For this endeavor they introduce a unified framework which treats all text-based language problems as *text-to-text* problems, which takes only texts as input and producing texts as output. The task is specified by a special token which prefixes the input sequence. This enables to directly apply the same model, objective, training procedure, and decoding process to all examined tasks. The authors created the *Text-to-Text Transfer Transformer (T5)* to examine existing techniques and do not propose new methods. To train the model they also created a new dataset called "Colossal Clean Crawled Corpus" (C4) by cleaning up Common Crawl's web extracted text and filter out any pages that were not classified as English. The final dataset is about 750 GB in size. The *T5* model is an encoder-decoder Transformer very close to the original one with the modifications of removed Layer Norm bias, layer normalization placed outside the residual path, and using an alternative position embedding scheme. The configuration of encoder, respectively decoder is similar to *BERT-base* and the text is encoded by *SentencePiece*. [25] The encoder and decoder share the same weights, which do not affect computation but halves the parameter count. The most effective training objective they used, was corrupting spans like *SpanBERT*. Also the authors found training longer (more steps), rise the batch size and increased parameter count have all positive impact on model performance. Bigger datasets are also beneficial as the fewer repeated data the model is trained with, the better the performance gets. Furthermore, pre-training on unsupervised data worked as well as pre-training on a multi-task mixture of unsupervised and supervised tasks before fine-tuning. Additionally, fine-tuning was most effective when all parameters of the model where updated jointly without parameter (un-)freezing, albeit this is the most computational intensive strategy. [38]

ELECTRA

Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA) is an LM, which introduces a more sample-efficient pre-training task than MLM, called *replaced token detection*. For pre-training, the input is corrupted by replacing

some tokens with plausible alternatives, sampled from a generator network. The actual LM is trained by detecting if the tokens of the input sequence are replaced or not. So the sample efficiency of this method is higher than MLM, because the task is defined over all input tokens instead just over the masked ones, which boils down to typically just learning from 15 % of the tokens per sample by models like *BERT*. Also no *[MASK]* tokens are needed, which prevents the pre-training/fine-tuning discrepancy. This method is supposed to be especially effective for small models and the method can be used to either train a model with much less compute or with the same compute and getting a better performing model. E.g. to reach the performance of *RoBERTa* and *XLNet*, just 1/4 of the compute is required for *ELECTRA*. Furthermore, *ELECTRA* is also more parameter-efficient, as smaller models reached or surpassed the performance of much larger models.

Although the generator/discriminator approach is reminiscent to *GANs*, the generator produces no adversarial tokens and is trained with maximum likelihood. The architecture of the discriminator is mostly the same as *BERT*, and the used generator is an MLM about 1/2 or 1/4 of the discriminators size, but the generator can be any model that produces an output distribution over the tokens. [9]

2.2 Recommender Systems

Recommender Systems are software tools using algorithms to suggest recommendations to a user. Generally, RS serve a big audience of users and are available as online services as part of other online services like e-commerce websites, music, and video streaming platforms or news websites. To make personalized recommendations possible, RS need information about the users, either as explicit or implicit feedback.

Explicit Feedback The users provide information directly. This can be user profile information like age, gender, origin, interests etc. Also direct feedback on items like ratings, likes or reviews fall under this category.

Implicit Feedback This category subsumes all traces of behavior patterns that users show. This can be clicks on items, buys of items, the time a user engages with an item like how much of a video, music track or news article a user consumed. Click streams on websites count as well as every piece of data a company collect due to the use of their online service.

Generally, the data a RS deals with are a set of users U and the set of items I. Whereby all user data like profile information, session data and pageviews are associated with the user u and all item data like content, description and metadata are associated with the item i. Furthermore, it is distinguished between *observed* and *unobserved* user-item interactions. Where *observed* interactions describe all interactions where a user has engaged with an item like a pageview or a like/dislike and *unobserved* describes all interactions between user and items which do not took place yet.

Machine Learning (ML) based RS need *objective functions*, which will guide the learning process. Usually two kinds of objective functions are used, either a function for a classification task or a ranking task. Where the classification function measures how well a RS can predict the items a user will buy, read, watch etc. in the future, with no ordering. On the other hand, the ranking function measures how well a RS can predict an ordered list of items where higher ranked items will match the users interest more than lower ranked.

Some RS, which operate on a very large corpus of millions of items, divide the task into subtasks to make the problem tractable. The systems typically have a *candidate generation* stage, which pick a few hundreds/thousand items from the large corpus which may interest the user. This stage typically uses less features for computation reasons and is a classification task, whether the user is interested or not. Next comes a *scoring* stage, which scores the items for user interest and ranks them into an ordered list. This stage can utilize more features as the input is much smaller, which reduces computation costs a lot. The last stage may be *Re-ranking*, which takes additional constraints for the final ranking like diversity, freshness, and fairness into account.

General Methods

Collaborative Filtering (CF) The assumption in CF method is, that users with similar interests in the past will also have similar interests in the future. If one user is interested in an item, another user with a similar history will probably be also interested in that item. So the similarity of user interest is calculated trough the similarity of the rating/view history of the users. Therefore, CF is also referred by [45] as "people-to-people correlation". Through a high amount of user-item interactions, the users collaborate with each other. This allows pure CF methods to work without any knowledge about the items. To yield a good recommendation performance an as big as possible amount of

users must be available. For recommendations with CF a user history must be present. New users and items have to be learned and the recommendation model has to be updated or trained from scratch. This can be circumvented by combining CF with Content based in a hybrid model and using deep learning models. [16] In CF a distinction is made mainly in two methods: *Neighborhood* and *Latent Factor Models*. *Neighborhood* methods are focused on the relationship between users or items. Whereas *Latent Factor Models* transform the users and items into the same latent factor space. These factors characterize the users and items and can be used to compare the users interests with the characteristics of the items. [42, 21]

Content based Contend based systems learn to recommend items, which are similar to that of a users history. The similarity measure is based of descriptions/features associated with the items and profiles, which assigns importance to these characteristics. These are either manually created or automatically extracted. Thus recommender systems research, especially with the contend based methods, are strongly rooted in the field of *information retrieval* and *information filtering*. The recommendation is based on how similar the user profile and the item features are. The profiles are generated through analysis of user behavior, user feedback, and information the user provides directly. For a good precision no big user group is mandatory but a user history must be present. Moreover, new items can be recommended without new model training. [42, 21]

Knowledge-Based Knowledge based recommenders are based on available, detailed content like technical or qualitative properties. These usually have to be provided manually and have to be available for items and users. Here typically a user interaction is mandatory to generate the user profile. An example are *constraint-based* recommenders, which suggest items by properties like price, category or genre. These systems are especially useful if no user history is available. [42, 21]

Hybrid Systems In hybrid systems, different methods are combined to compensate for weaknesses of one method or to integrate more information to yield better recommendation performance. [42, 21]

2.2.1 Deep Learning

Deep Learning methods are relatively new in the field of RS. Here, Deep Neural Networks (DNN) are used to suggest recommendations. Due to their non linear computations, very complex relationships in the data can be modeled in contrast to linear models like *Matrix Factorization*. They are typically used to implement CF or Content based methods and often combine several methods to a hybrid system. DNN are also useful to retrieve information from different sources like text, pictures, videos, music, categorical and many other data modalities.

There is a broad collection of neural networks, which are utilized for current RS like *Multi Layer Perceptron* (MLP), *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN) and its specializations *Long Short Term Memory* (LSTM) and *Gated Recurrent Unit* (GRU), and recently *Transformer* based neural networks. A comprehensive description of these neural networks is given by [20], except for *Transformers* which is described in [53]. A good overview over deep learning based systems are given by [68]. [55] also covers DL Systems, albeit with a focus on session-based recommendations. In the course of this section, some selected works are presented, which are representative of the concerned *NN* architecture, are especially noticed by the RS research community or use interesting methods. Some other *NN* architecture like *Graph neural Networks (GNN)* or methods like *Deep Reinforcement Learning (DRL)* are also utilized by RS but will not be covered by explicit examples as they are less broadly used and would go beyond the scope of this work.

Transformer Based RS

RS with the most recent type of *Artificial Neural Network* (ANN) architecture, *Transformers* [53], are very sparse compared to traditional DL based recommenders but gain traction. Examples are [7, 3, 66, 24, 48, 63, 67, 70, 6]. The adoption of *Transformers* in the RS field is also underlined by [47] with the creation of the *Transformers4Rec* library to make the usage of *Transformers* in RS more accessible. Some of the more noticed *Transformer* based RS are [24, 48, 7], which are now briefly introduced as representation of this kind of RS.

Self-Attentive Sequential Recommendation *Self-Attention based Sequential Recommendation model (SASRec)* uses the decoder module of a standard *Transformer* model for recommendations with just a few changes. The model is a causal *Transformer*, which attends only to previous tokens like *GPT*. The input to the model is the user history per sample. The item IDs in the user history are the input tokens for the model, like words are input tokens for a LM. First, the items are fed into an embedding layer, which learns an embedding for the item and the position. The difference to the standard *Transformer* is the learned position embedding, which performs better than the fixed position embedding. Next, the embeddings are processed by stacked self-attention blocks, with the difference that the authors used *single-head* instead of *multi-head* self-attention, which performed better in their experiments. The next item prediction, given the first t tokens, is produced with a *Matrix Factorization (MF)* layer with shared weights with the input item embeddings. This essentially boils down to computing the inner product of the next output token embedding with all item embeddings and ranking the results by the computed score. For training the authors adopt the binary cross entropy loss to make the score between the output token embedding and the expected item embedding higher than the score for a randomly sampled negative item embedding. [24]

BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer *BERT4Rec* adopts the *BERT* LM to the RS domain and hypothesizes that the users historical behavior may not always follow a rigid ordered sequence due to various unobservable external factors, and that the power of hidden representations for items will benefit if they can encode the information from left and right context. Thus, unidirectional left-to-right models do not exhibit the full potential to model optimal representations for the user behavior sequence. It uses essentially the same architecture as *BERT* inclusive the *cloze/MLM* task but utilize learned embeddings, remove the next sentence prediction task, and segment embeddings since the authors model the user behavior as only one sequence. Additionally, the model is used in an end-to-end fashion without pre-training, as knowledge is not transferable between datasets with different items. For the output, a two layer feed-forward network is used, whose output is multiplied with the shared item embedding matrix, and a final softmax layer assigns probabilities to the items. The training objective is a negative log-likelihood of the masked targets which maximizes the probability of the output embedding to be the actually masked item. The masked items training create a mismatch between the training and the final sequential recommendation task. This is addressed by inserting

the mask token at the end of the user's history and predict the next item based on the output embedding corresponding to the mask token. The experiments show that the bidirectional context of the model significantly improves its performance as well as the *cloze* task. [48]

Behavior Sequence Transformer for E-commerce Recommendation in Alibaba *User Behavior Sequence Transformer (BST)* models the recommendation task as Click-Through Rate (CTR) prediction problem, using a *Transformer* to take advantage of the behavior sequence of users. *BST* is build upon the *Wide and Deep* model by concatenating miscellaneous features and sequential features from the user behavior and feed them into an *MLP*, which predicts the click probability of a target item. The miscellaneous features like user profile, context, item, and cross features are embedded and concatenated into one vector. The items from the user history are represented by the embedding of their features like *item_id* and *category_id*, which are concatenated with a positional embedding into a single item representation vector. The item representations are processed by a *Transformer* layer, whose output is fed into the final *MLP*. [7]

MLP Based RS

Neural Collaborative Filtering With *short for Neural network-based Collaborative Filtering (NCF)* the authors introduce a general framework with *General Matrix Factorization (GMF) MLP (NCF)* and *Neural Matrix Factorization (NeuMF)* as instances of that framework. They focus on *CF* on the basis of *implicit feedback* and thus propose a probabilistic model, which predict the interaction probability of a user and an item. The input to the model is a one hot encoded vector for users, respectively items. These input vectors get transformed into dense embeddings, which can be seen as user (item) latent vectors in the context of a latent factor model. Next, the embeddings are fed into a multi-layer neural architecture which maps the latent factors to prediction scores. The model is trained by minimizing the binary cross-entropy loss between the predicted probability score of the model and the ground truth. Negative instances are randomly sampled from unobserved interactions.

In the *GMF* instance the element-wise product (Hadamard product) of the user and item embeddings are projected trough a learned weight matrix and sigmoid as the activation function to produce the final output score. In the *MLP (NCF)* instance, the user and

item embeddings are concatenated and fed trough an *MLP*, which learns the interactions between users and items and outputs the probability score trough a sigmoid. This endows the model with a large level of flexibility and non-linearity to learn the user item interactions compared to an *MF* model. The *NeuMF* model combines *GMF* and *MLP (NCF)* into an ensemble model, which leverages the linearity of *MF* and the non-linearity of *MLPs*. The overall architecture of *NeuMF* is shown in figure 2.9. [16]

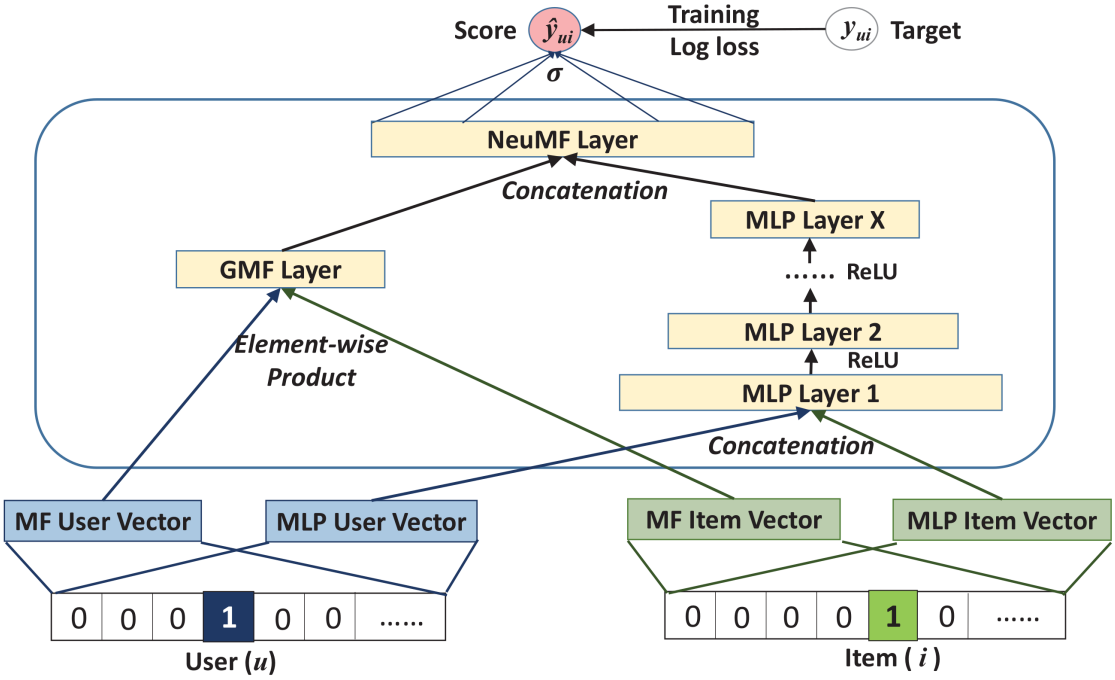


Figure 2.9: NeuMF model: On the **Left** the *GMF* layer can be seen. On the **Right** the *MLP* instance is shown. The layer on the **Top** combines the final embedding layers of both models and yield the final score. [16]

Wide & Deep Learning for Recommender Systems In the *Wide & Deep learning* framework the authors combine a wide linear model and a non-linear deep *MLP*. The joint training of both models leverage the memorization capabilities and effectiveness of cross-features of the linear model under sparse and high-rank user-item interaction conditions and the generalizeability to unseen feature combinations and the need for less feature engineering effort of deep networks. Memorization in the wide model is achieved by learning frequent co-occurrence of features or items in the historical data. The input to the wide model is raw input features and transformed features like cross-product transformations, e.g. "AND(gender=female, language=en)". The sparse input features

for the *MLP* are converted into embeddings and concatenated with dense input features and fed into the model. The final prediction of the wide and deep part are the weighted sum of their output log odds. [8]

CNN Based RS

Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding The *ConvolutionAl Sequence Embedding Recommendation Model(Caser)* is a *CNN* based model, which represents the behavior sequence of users as "images" of ordered items (rows) x latent dimensions (columns). This embedding matrix is searched with various convolutional Filters for sequential patterns by modeling them as local features and thus do not model sequential behavior as adjacent actions. These filters are applied vertically and horizontally to capture point-level, union-level and skip behaviors. Where at *point-level* the target action is only influenced by the previous action, at *union-level* the target action is jointly influenced by ordered previous actions and *skip behaviors* where the impact of previous actions skip some steps between these and the target action. *Caser* is made up of three components, *embeddings*, *CNN* and *MLP*. The item input to the model is translated into embeddings, which are learned together with the other components. Beside the sequence embeddings, the model also maintains an embedding for user features. The input to the CNN is the stacked embedding matrix $E \in \mathbb{R}^{L \times d}$ with L items of d dimensional embeddings. The *Horizontal Convolutional Layer* captures union-level patterns with multiple union sizes by sliding filters of various heights and full width d from top to bottom and thus interacting with all successive items. The *Vertical Convolutional Layer* captures point-level sequential patterns through weighted sums of previous item embeddings by sliding filters of dimension $L \times 1$ from left to right, where each filter acts as a different aggregator. To get more high level abstract features, the concatenated output of the *CNN* layers is fed into an *MLP*, whose output represents the sequence embedding. The sequence embedding is concatenated with the user embedding and gets projected into the output layer. For training, the output is transformed into a probability by the inner product of the output layer and the target item with a final sigmoid. The training samples for *Caser* are generated by a sliding window of $L + T$, where L successive items are the input and T successive items are the target, whereby each window produces one training sample. *Skip behaviors* can be considered by skipping the direct target item and replace it with the next target items. The model learns by minimizing the binary cross-entropy with three randomly sampled negative items. [51]

DKN: Deep Knowledge-Aware Network for News Recommendation The *deep knowledge-aware network (DKN)* is a model, which incorporates external knowledge from a *Knowledge Graph (KG)* into the sentence representations from news titles and use an attention mechanism over a users history to predict the users interest in a candidate news article. The key component is a *knowledge-aware convolutional neural network (KCNN)* with multiple channels and word-entity alignment to fuse the semantic and knowledge-level representations of news articles. The words in a news title are associated with a relevant entity from the *KG* and its contextual entities. The *KCNN* treats word, entity and contextual entity embeddings as multiple channels like colors in an image. To eliminate the heterogeneity of the word and entity embedding spaces the entity embeddings are projected into the word embedding space with the same size trough a linear or non-linear learnable transformation function. The news and candidate news articles are transformed into an embedding by the *KCNN* to which an attention module is applied. This dynamically aggregates the users history by matching viewed articles with different weights to the target article. Specifically, two article embeddings are fed into an attention *DNN*, which assigns an impact weight to the article. This is done for all viewed articles, whose weights are normalized trough a softmax, which is used to calculate the weighted sum of all viewed articles to generate the user embedding. The final prediction is computed by an *MLP*, which is fed with the user embedding from the attention module and the target news embedding. [54]

RNN Based RS

Session-based Recommendations with Recurrent Neural Networks This work is a seminal paper, which proposed *GRU4Rec*, an *RNN* based RS for session-based recommendations. As input, the network is fed with a sequence of one-hot encoded items, which was superior to embeddings for this model. The model consists of possible multiple *GRU* layers with skip connections from input to intermediate *GRU* layers and an optional *MLP* for the output, while the authors achieved best performance with just one *GRU* layer without *MLP*. The output of the model is the most probable next item of the input sequence. The model is trained by minimizing a ranking based loss where *TOP1* and *Bayesian Personalized Ranking (BPR)* showed the best performance, whereby *TOP1* was devised for this model and is the regularized approximation of the relative rank of the relevant item. To train the model, mini-batches from different sessions are constructed where each batch consists of the current event of the corresponding session the next even

of the session as target. The next batch is constructed from the next event and next target. As sessions have different length, on the place of an exhausted session the next unprocessed session is used and the network state for the session is rest. Negative samples for the ranking based loss are drawn from the sessions of the current batch, which corresponds to popularity-based sampling, because the likelihood of a sample to be in the other sessions is proportional to its popularity. [18]

Additional improvements to the original algorithm is made by the author in [17]. Firstly, improved loss functions *TOP1-max* and *BPR-max* are proposed, which tackle the vanishing gradient problem when having many negative samples with a low score. This is achieved by a weighted sum of the negative example scores with a softmax, which assigns high scoring samples more weight. Further, they improved sampling by also sample from the whole dataset in addition to sampling from the sessions in the batch, because high scoring negative examples are crucially for the model to learn efficiently and the batch of sessions is too small to draw many high scoring samples. The additional samples are drawn proportional to the support of the item with a hyperparameter to slide between more uniform and more popularity based samples. Additionally, the use of embeddings which share weights for the input and output embedding show further minor improvements.

Embedding-based News Recommendation for Millions of Users The authors propose a news RS with three components in an end-to-end manner. First, the articles are converted into a distributed representation, a.k.a. embedding, based on a variant of *denoiseing autoencoder (DAE)*, where the output of the encoder represents the article embedding. Next, the article embeddings are fed as sequences, representing the users history, into an *RNN* to generate a user representation. Lastly, candidate articles are matched to the user by taking the inner product of the candidate article and user representation and ranked by score and additional factors like freshness and expected number of page views. Importantly, the candidate articles are de-duplicated to not dissatisfy the user with many different articles on the same content. This is done by skipping articles whose cosine similarity with other higher scored articles surpass a threshold. The *DAE* is trained with the elementwise cross entropy objective with an additional loss, which ensures that articles in similar categories have more similar representations than articles of different categories. The noise is produced by masking the input tokens with stochastic corruption and corruption rate p. For the user representation a single layer *RNN* is used, whereby a *GRU* performed slightly better than an *LSTM*. The user representation

component is trained with a log loss by forcing the relevance score of all clicked articles in a session to be higher than all shown not clicked articles. The relevance score is simply computed by the inner product of the user and article embedding. [32]

Attention Based RS

MARS: Memory Attention-Aware Recommender System The *Memory Attention-Aware Recommender System (MARS)* model tries to model the diverse interests of users by using a memory component of the user's history and an attention mechanism to dynamically adapt the user representation to a candidate item. The attention mechanism also has the effect of greater interpretability to explain the recommendation. The user memory component is built by computing an embedding of all items liked by a user through a *CNN* and concatenate the embeddings to a single memory component vector. The item embedding is generated from the textual description of an item. First, a word embedding layer is utilized, which produces the embedding matrix. This matrix is processed by one *CNN* layer with several filters, which span the whole embedding dimension and a window size c. From each filter the most important feature is extracted through max-pooling and a final fully connected layer projects the features in the item embedding dimension. The candidate item embedding is generated the same way as the item embedding, only with differing inputs and weights. The dynamic user representation is computed with the attention mechanism. Therefore, an attention vector is obtained through a softmax of the inner product of the memory component vector and candidate item. The user representation is the weighted sum of the memory components by the attention vector. The final score of the candidate is given by the inner product of it and the dynamic user representation. The training of the model is inspired by *BPR* and done by pair wise ranking loss of the relevance scores of a positive and negative sample of user liked items, while user, positive and negative items are uniformly sampled. [73]

2.2.2 News Recommendation

News recommendation comes with important properties, which have to be taken into account. News articles are very time-sensitive with a quickly expiring relevance and out-of-date news are substituted with newer once frequently. People are topic sensitive and usually interested in multiple specific news categories, so an RS must dynamically extract the current user's interest from a diverse reading history. News texts are highly

condensed and comprised of a large amount of knowledge entities and common sense. [54] Thus, it is important to understand the content of articles, the user preferences and select articles individually for users based on content and preference. Also, the RS have to be scaleable and must respond with a very low latency to every user access. [32] Additionally, usually no explicit user feedback like reviews, ratings or likes is available, so the personal user interest has to be inferred from his implicit feedback like clicks. [58] Further challenges in news recommendation are described by [12] and summed up in table 2.1.

Collaborative Filtering Approach Challenges	**Content-Based Approach Challenges**	**Hybrid Approach Challenges**
Cold-start Problem	Recency	Data Sparsity
Explicit User Feedback	Continuous Changes in News Items Set	Scalability
Continuous Changes in User Interests	Unstructured Content	Serendipity Problem (recommending same item again)
Changing User Interest due to Topic Divergence	Response Time	Fraud
User Modeling or Profiling	News Recommendation from Multiple Sources	
User's Privacy Problems	Cross Lingual News Recommendations	
Gray Sheep Problem		
The Context-Dependent Relevance of Items		
Unpredictable Behavior for the Same Items		
Unwillingness of Users to Register		
Using Shared Devices		

Table 2.1: Challenges in news RS collected by [12]

An example workflow of personalized news recommendation is depicted by figure 2.10. First, the user visits a news platform. Second, the RS has to recall candidate news from a large-scale news pool. Third, the candidate news are ranked by the RS, which may include criterions like relevance, recency and filtering by e.g. de-duplication. Forth, the narrowed down top k list of news is displayed to the user. Lastly, if the user chooses an article, the user interaction is logged and used to update the user profile. [58]

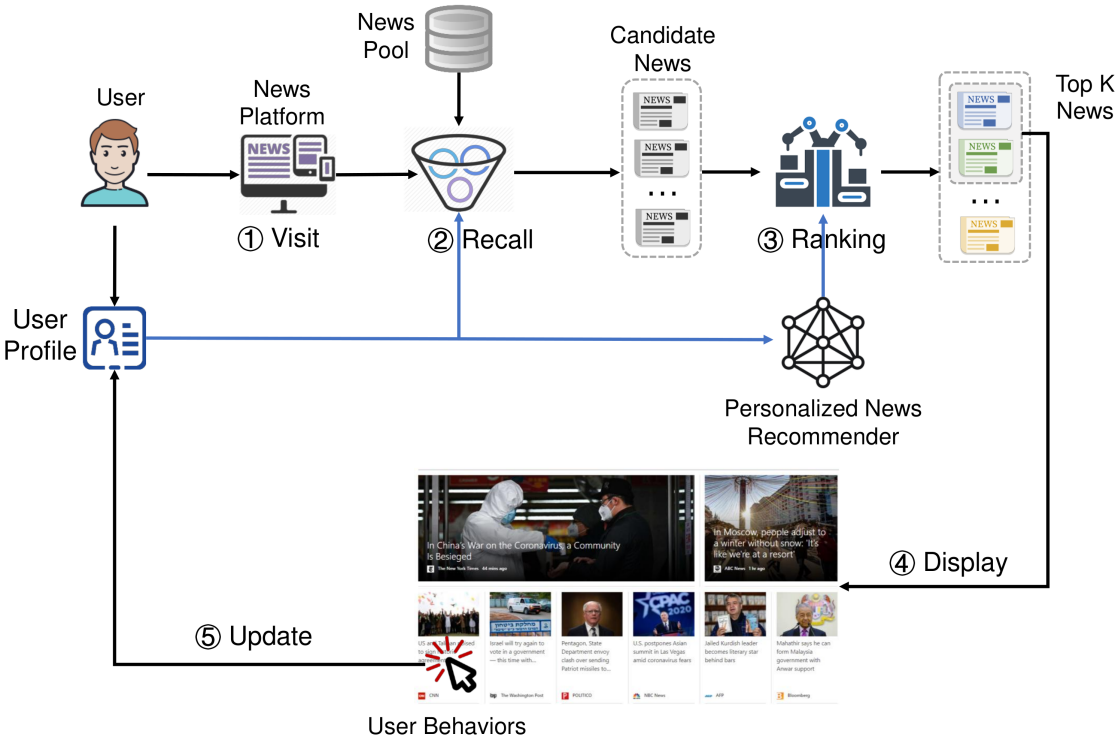


Figure 2.10: "An example workflow of personalized news recommender systems." [58]

In figure 2.11 a common personalized news recommendation model development framework is shown with key problems to solve. In news recommendation, the modeling of news is crucial, with the core problem to understand content and characteristics of news. To understand the personal interests of users, a user model has to be developed to infer their interests based on user profiles, including behavior and other user characteristics. Now, the RS have to rank candidate news based on the user and news representations generated by the news and user model with certain criteria like relevance and freshness. The next step for a well performing RS is a proper model training with defining objectives as a critical part. Lastly, the model has to be carefully evaluated. [58]

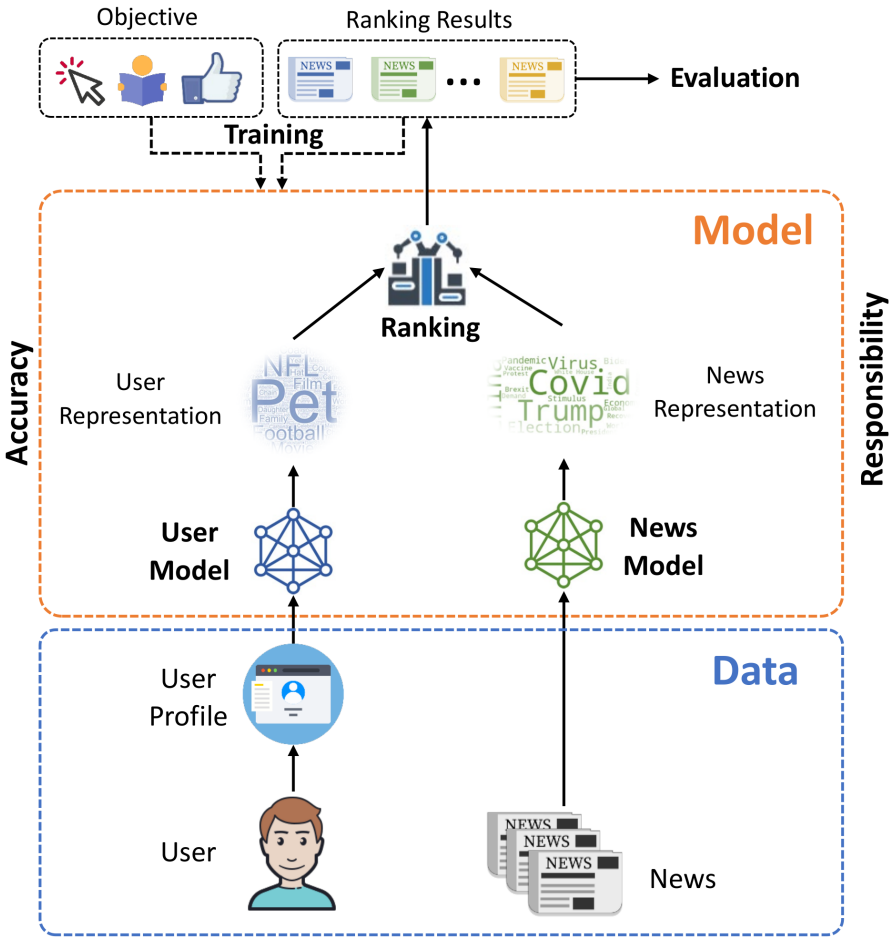


Figure 2.11: "A framework of the key components in developing personalized news recommendation model." [58]

User modeling in news and general recommendation, like e-commerce or movie recommendation, have a close relation. The core *NN* architectures, like *CNN*, *RNN* and *self-attention*, are often the same and widely used in news and sequential recommendation. Despite related techniques and methods, there are unique characteristics in user modeling for personalized news recommendation [58]:

Short news lifecycles Interactions with news typically span only a few days in comparison with e-commerce, where items can be relevant for years. So ID embeddings are often not very useful to represent news items and it is better to use the content for learning a representation. Also, the exploitation of collaborative signals in user

modeling is limited due to the quick vanishment of old news and the large fraction of cold news in the inference stage.

Fine-grained candidate-aware user modeling The rich content and context information of news can be leveraged to model fine grained interactions (e.g. word-level interactions) between user behavior and candidate news instead of just overall item embedding with user behavior.

User modeling as document modeling News contain rich text and user modeling can be reformulated as document modeling where texts of clicked news are embedded in user "documents". *PLM* can be utilized to enhance user modeling and shows the proximity of *NLP* and news recommendation.

Potential strong temporal diversity preference Users prefer diverse news and other than in general recommendation, the next clicked news tend to be different from the previous one. A recent study [60] suggests that news modeling should not be considered as a standard sequence recommendation task, which is supported by the success of standard self-attention models [48] over causal self-attention [24] and other sequential models, as they focus more on global context then sequential dependency.

Some of the reviewed works already concern the news RS domain. Where Okura et al. of section 2.2.1 modeled news content with a *GRU* based *AE* and the users as a sequence of article embeddings likewise with a *GRU*. Wang et al. of section 2.2.1 used a *CNN* combined with entity embeddings of a *KG* to model the news and an *Attention Network* to model the user based on clicked news embeddings and candidate news embeddings. Other approaches are taken by [72], [66] and [59]. Zheng et al. [72] proposed a *Deep Q-Learning* based recommendation framework, which is a *DRL* based method and can model future rewards explicitly, thus keeping track of current and future rewards simultaneously where the reward is determined by user clicks and activeness. The model predicts the click probability for candidate news with a *multi-layer Deep Q-Network*. The *Q-Network* essentially models the user behavior and is divided into two networks, which represents the *value function* and *advantage function*. Due to the nature of *RL*, they are able to do online recommendation with exploration and actively explore the user's interests instead of just learning from historical data. For exploration, they use an additional exploration *NN* and a *Dueling Bandit Gradient Descent (DBGD)* method to choose between recommendations from normal and exploration *NN*. This leads to more diverse recommendations while not hurting accuracy too much in contrast to random exploration. Additionally, to click streams, they use user return patterns

and maintain an activeness score per user as a supplementary signal to capture more user feedback information. As input, they do not use the news content but one-hot encoded features, which describe whether a certain type of property appears in the news, context of the pageviews like time, and user features like properties of previous clicked news. Initially, the model is trained with recorded offline logs and in the online phase, the model learns by interacting with the users. Zhang et al. [66] propose *User-News matching BERT (UNBERT)*, which achieves the first rank in the *MIND* leaderboard.[3] at the time their paper was published. *UNBERT* uses a pre-trained *BERT* model to obtain a news representation from news titles and adopt the *pretrain-finetune* strategy to fine-tune the whole model to the in-domain news data to generate scores for all candidate news by predicting their click probabilities. It captures multi-grained user-news matching signals at word-level and news-level using a *Word-Level Module (WLM)* and *News-Level Module (NLM)*. The candidate news as "News Sentence" and the sequence of user clicked news as "User Sentence" is both fed into the same *BERT* model by separating them with the special *[SEP]* token and the concatenated words of clicked news titles with the *[NSEP]* token. The word level news representation is given by the embedding of the *BERT* typical *[CLS]* token. The news level representation is generated by aggregating the word embeddings of each news with an *Attention Aggregator* and process them with the same *Transformer* layers from the *WLM*. The *NLM* representation is likewise given by its *[CLS]* token. The final click prediction is given by an *MLP* with the *WLM* and *NLM* representations as input. In this model, the news and user representations are tightly coupled and not modeled with separate modules. Wu et al. [59] achieved the highest rank on the *MIND* leaderboard at the time of writing this thesis. They explore the effectiveness of combining different *PLM* for news representation with existing methods like Okura et al. of section 2.2.1, which improved the recommendation performance significantly. The best results are achieved by combining the *UniLM PLM* by [2] for news representation with an *attention network* for pooling word embeddings and the model from [57]. The model is modified by exchanging the standard *Transformer* layers with the *Fastformer* [61]. So the user is modeled by feeding the *PLM* encoded news articles into the *Fastformer*, whose final news embeddings are pooled by an *attention network*. The click probability score is obtained through the inner product of the user and news representation vector. For a comprehensive overview over personalized news recommendation the recent survey by [58] can be consulted.

[3] https://msnews.github.io/

2.3 Conclusion

As shown in the analysis so far, a successful news recommendation system has to meet different criteria and the user modeling and news modeling module are crucial components. To establish a baseline for the news recommendation dataset, a relatively simple model with an extendable architecture will be chosen. The model should be trainable in a preferably short period of time for easy reproduction and to be able to rapidly perform experiments and test prototypes. Also, the RS have to meet low latency and resource consumption criteria to make in production serving to users possible. The system has to be extendable, to use the baseline as starting point for further experiments, which may incorporate advanced user and news modeling. To not relay on pre-trained models whose training parameters are not under own control, the baseline will not incorporate *PLM*. Recently, *Transformer* base models have shown to be performant but also very sensitive to datasets and hyperparameters, so these types of models will not be chosen for the baseline. Although *RNN* based models have shown good performance in RS, due to their sequential nature they are difficult to parallelize and may have too expensive training time.

A previous project [29] has shown that the *CNN* based *MARS: Memory Attention-Aware Recommender System* from [73] has several advantages, which qualify it for the baseline. Benefits on the technical site are a fast training with very few training epochs to converge, insensitivity to batch sizes and a good parallelizability with many GPUs due to the *CNN*. The insensitivity to batch sizes is important as the data is high dimensional with large deviations in the view history of users and word count per news article. This leads to the need for much padding in large batches, which consumes a very big proportion of the computational and memory resources. Alternatives for using large batches are *Ragged Tensors*[4] or sorting the training data by length of user history and articles, which both have also downsides. According to [29] *Ragged Tensors* have shown to be immature and thus of very limited use in *Tensorflow* for this kind of models and not implemented at all by *PyTorch*. The sorting of training data may have a huge negative impact on recommendation performance and may also lead to a poor data efficiency so the model needs to train on more samples, which compensate or overweigh the gain in data throughput.

[4] A kind of nested arrays of different length also called jagged/ragged array, see: https://en.wikipedia.org/wiki/Jagged_array

The *MARS* model meets all criteria for a news recommendation system. It is able to distinguish the user's diverse interest in possible multiple specific news categories through the *deep adaptive user representations*. The model understands the news content with news modeling using a *CNN*. The user's preference is modeled with a memory component and recommended articles are selected based on the candidate articles content and the user's preference. Also, the model needs no explicit user feedback and infers the user's interest from the implicit feedback of clicked articles. The attention mechanism which models the user's diverse interest through *deep adaptive user representations* is a unique feature to the best knowledge of the author. The model attends to a context item, which can be for example the currently read news article, and derives a user profile dynamically adapted to the context item. Moreover, the representation of the items and the user profile is trained and optimized jointly in an end-to-end fashion, whereas many other RS models train different parts of the model separately. This simultaneous training of the news and user model makes the training pipeline less complicated and promise a good tuning of the components with each other without manual tuning of the individual model to each other. Furthermore, the neural network topology is easily implementable for its relative simplicity and its architecture is easy adaptable to different recommendation scenarios and types of context data. A drawback of the model is, that it does not consider the sequential ordering of the view history. This drawback is mitigated, since the different parts of the model, e.g. the CNN and Dense networks, are straightforward to replace with more powerful and sophisticated Neural Networks, like *PLM* for news modeling and *Transformer* based networks for user modeling with consideration of sequential/temporal features. Additionally, through the use of standard CNNs and basic dense networks, the speed to yield a recommendation for a user should comply with the strict latency requirements in a real world recommendation scenario. Furthermore, the architecture is suitable to adapt the user profile to a live stream of user interactions without the need to precompute user profiles. Finally, as the model has its origin in e-commerce recommendation, the implementation can help to answer the question of challenges in the adaptation to the news recommendation domain.

So far with this analysis some questions have already been answered and can be summarized as follows:

Q1: After thoroughly analyzing the requirements of a news recommendation system and exploring DL based RS techniques in different directions, a model for the baseline has been chosen. This is the first part to fully answer the question, which will be continued in the next chapter.

Q4: The challenges in news recommendation are addressed in section 2.2.2, which states, that news modeling is one of the most crucial parts in news recommendation. A framework to develop news RS is shown in figure 2.11 and a listing with special news RS charactteristics in distinction from other RS is paraphrased in listing 2.2.2. Practical challenges are discussed in the next chapter.

Q5: In section 2.2.1, some models are introduced, which incorporate different features from items and users which can generally be used to incorporate different metadata into a (news) RS. E.g. the *BST* model embeds miscellaneous categorical features from items and concatenates them into one representational item embedding. Likewise, different features from a user are embedded and concatenated with sequential user history features and collectively fed into a final *MLP*. Also, the Wide & Deep model uses embeddings for categorical features and concatenates them with dense features, which are the input for an *MLP*. Also, the *Caser* model uses concatenation of different feature embeddings, which are jointly processed by an *MLP*. Also, the possibility to indirectly using metadata by applying it in filtering in the pre-processing stage or using it as labels to be predicted by the model as auxiliary tasks are plausible.

In the next chapter the questions **Q2** and **Q3** are addressed and open points of **Q1**, **Q4** and **Q5** are approached.

3 Experimental Design

In this chapter, the experimental design is described, which is divided into dataset analysis, model, pre-processing and toolchain description. The dataset is introduced and analyzed to find useful metadata, which can be accessed to enhance the *RS* and to plan the pre-processing pipeline. Hereafter, the *MARS* model, which serves as baseline, is described, which was already briefly introduced in section 2.2.1. Since the model was implemented from scratch, as there is no implementation from the original authors available, its implementation and technical details are described. In a previous work [29], the paper from [73] was replicated and the reproduced model implementation of this work was validated through the use of public available datasets used in the paper. The three datasets used in [73] consist of user rated items where movies have a short synopsis as text data and e-commerce items have a textual description. In contrast to the aforementioned datasets, the news dataset has no user ratings but user engagement data and the textual data of the articles is the item content itself instead of a description of the item. Next, the pre-processing of the data is described, including data processing for the training and inference stage. Also, the adaptions of the whole RS pipeline from the e-commerce to the news recommendation domain is described. Lastly, the toolchain and the decision for the used tools over others are explained.

3.1 Dataset Analysis

The Dataset is kindly provided by *SCHICKLER Unternehmensberatung GmbH*[1] to support this research. Schickler work together with different publishers and *dpa Deutsche Presse-Agentur GmbH*[2] to enhance the users news reading experience. The dataset is fully anonymized and does not contain any personal data.

[1] https://www.schickler.de
[2] https://www.dpa.com

The dataset is composed of 313 565 551 pageviews from 56 199 311 users and 823 947 (+1 with no full text content) articles. This leads to a density (ratio of observed to total possible pageviews) of 0,000 677 171 %, which makes this dataset much more challenging than the datasets in the original *MARS* paper with 0,24 %, 0,037 % and 0,0128 % density. There are 47 fields for articles and 35 fields for the pageviews, respectively 54 and 63 when expanding the *record* fields. A selection of the fields of the pageviews and articles are listed in table 3.1 and 3.2 together with statistics about number of unique values, percent of not null values and minimum, average, maximum, and standard deviation of fields containing texts or numerical values. The latter statistics are concerning number of words in texts and values of numerical data, where boolean fields are treated as one and zero values. Tables with all fields can be found in appendix A.2.

The most important field in the *pageviews* table 3.1 are the *user_id* to identify the user and the *article_drive_id* to reference the corresponding article. The *publisher_id* and *portal_id* are likewise important to do recommendations only for the portal or publisher, whose service is accessed by the user. Another essential information is the *fraction_article_read* field, which indicates how much of the articles are read by the user. For the RS baseline, the fraction the user reads is accessed to filter out *pageviews*, which are actually not important for the user. For further refined models, the *geo* record field with geographic information like city, county and approximated geographic coordinates of the *pageview* could be very useful, e.g. to recommend articles of local interest. The *session_id* could be leveraged for session based recommendation, especially when the history of the user is very short. Session based recommendations are also a possibility for personalized recommendations when a user does not allow to be tracked, respectively delete browser cookies. Also, the *session_referer_medium* and *session_referer_source*, which say something about how the user came across the article, may hold very useful information for the recommendation. A future task will be to investigate for which fields a DL model is able to find patterns to enhance the personalized recommendations.

The most important fields of the *articles* table 3.2 are the *article_drive_id* to connect the article with the pageviews and the content fields *article_header*, *article_teaser* and *article_full_text*. These content fields are in focus of this work and the most important information to represent the articles, where the *article_full_text* also contains the text of the header and teaser. For future work, the *is_plus_article*, which indicates a paid content article and publishing and modification date could be beneficial information for the RS. Also, the *article_dpa_id* field is very interesting, as it could be used to connect the articles from this dataset with articles from the *dpa*, which have very rich metadata

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
user_id	string	56199311	100.00				
session_id	string	167711970	100.00				
session_referer_medium	string	7	100.00				
session_referer_source	string	87	36.64				
geo	Record						
useragent	Record						
publisher_id	string	10	100.00				
portal_id	string	17	100.00				
page_view_id	string	312908725	100.00				
user_type	string	3	100.00				
x_scroll_pct	double	101	99.95	0.00	98.66	100.00	8.73
y_scroll_pct	double	101	99.99	0.00	40.74	100.00	28.31
time_engaged_in_s	int64	1036	100.00	0.00	30.88	489825.00	53.06
article_type	string	3	60.53				
is_paywall	bool	2	92.39	0.00	0.26	1.00	0.44
content_type	string	5	100.00				
user_engagement_segment	string	7	61.90				
article_drive_id	string	823944	48.14				
fraction_article_read	double	79904	48.14	0.00	0.24	1.00	0.28

Table 3.1: Schickler News Recommendation dataset: Statistics about pageviews.

as examined by [28]. The only drawback is the relatively low proportion of about 12 % of articles connected to *dpa* articles.

In table 3.3 statistics are shown about pageviews per user, article, publisher, portal, and session. It can be seen that users have a relatively low average view history length with a big standard deviation. The histogram of figure 3.1 with pageviews per user shows that the majority of users just view one to two articles. This indicates that the current recommendation methods do not catch most first time users with interesting further articles. This also promises that, through good recommendations, the increase of just one pageview per user would have a massive impact on overall pageviews and time spent on the portal. The average number of just 183,21 pageviews per article also shows that most articles have a relatively niche audience, which would likely also benefit from a targeted promotion to users. This is supported by the high standard deviation and the histogram of figure 3.2, which shows that most articles have very low pageviews, well below 100 and just a few over 1000. Figure 3.3 shows the pageviews per publisher

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
publisher_id	string	10	100.00				
article_drive_id	string	823944	100.00				
article_header	string	755922	100.00	0.00	6.35	29.00	2.60
article_teaser	string	738363	100.00	0.00	24.08	1485.00	15.38
article_full_text	string	821750	100.00	0.00	398.35	30642.00	279.24
is_plus_article	bool	2	97.77	0.00	0.19	1.00	0.39
article_dpa_id	string	47290	11.92				
is_dpa	bool	2	100.00	0.00	0.12	1.00	0.32
published_at_local	timestamp[us]	542925	100.00				
modified_at_local	timestamp[us]	736714	100.00				

Table 3.2: Schickler News Recommendation dataset: Statistics about articles.

and its portals. It can be seen, that there is a big deviation between publishers and especially portals in the amount of generated pageviews. All portals and particularly the small portals should benefit greatly from the aggregated amount of pageviews when it comes to train an RS, since more data is generally very beneficial for training a DL model. This also brings the pre-tain and fine tuning technique into play. Further work should investigate if training on the whole dataset with subsequent fine tuning of the portal/publisher specific data improves the RS performance.

Data	Unique	Pageviews			
		min	avg	max	Std.
PV per users	56 199 311	1	5,58	59 161	59,20
PV per article	823 947	1	183,21	4 007 936	6775,87
PV per publisher	10	11 421 549	31 356 555,10	73 769 662	20 521 363,43
PV per portal	17	475 987	18 445 032,41	73 769 662	21 215 710,41
PV per session	167 711 970	1	1,87	173	2,23

Table 3.3: Schickler News Recommendation dataset: Statistics about pageviews (PV) per group.

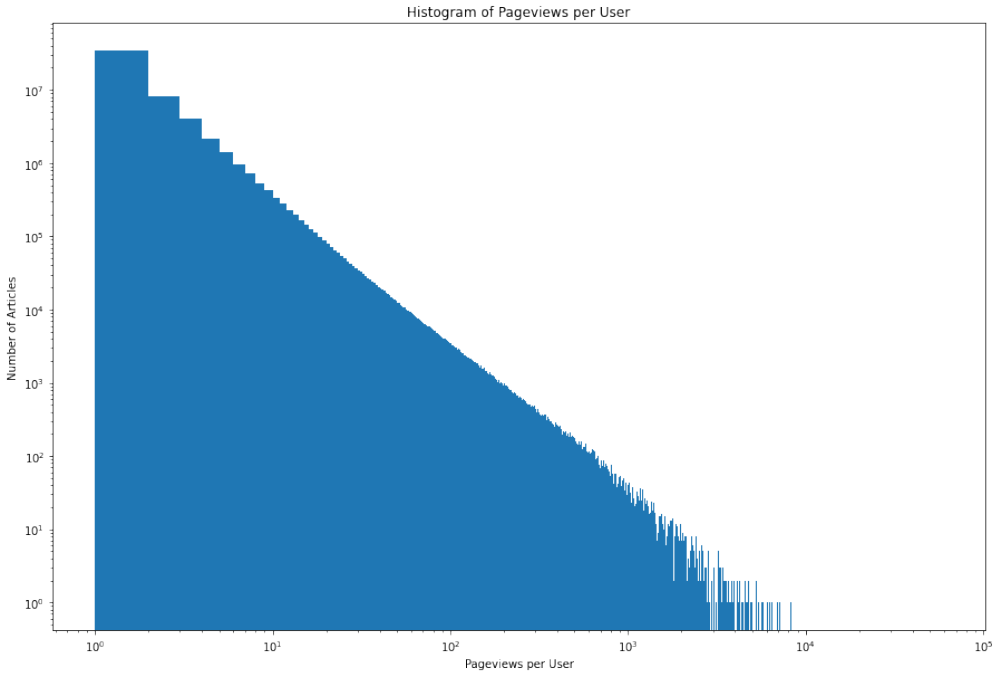


Figure 3.1: Histogram of pageviews per user with 59 161 bins and log scale on both axes.

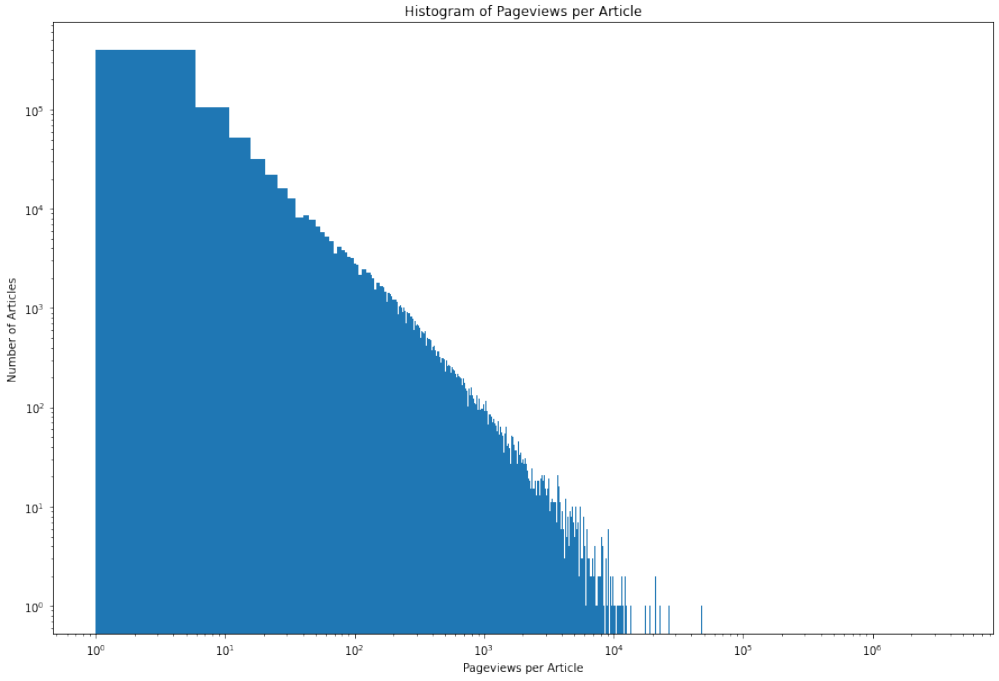


Figure 3.2: Histogram of pageviews per article with 823 948 bins and log scale on both axes.

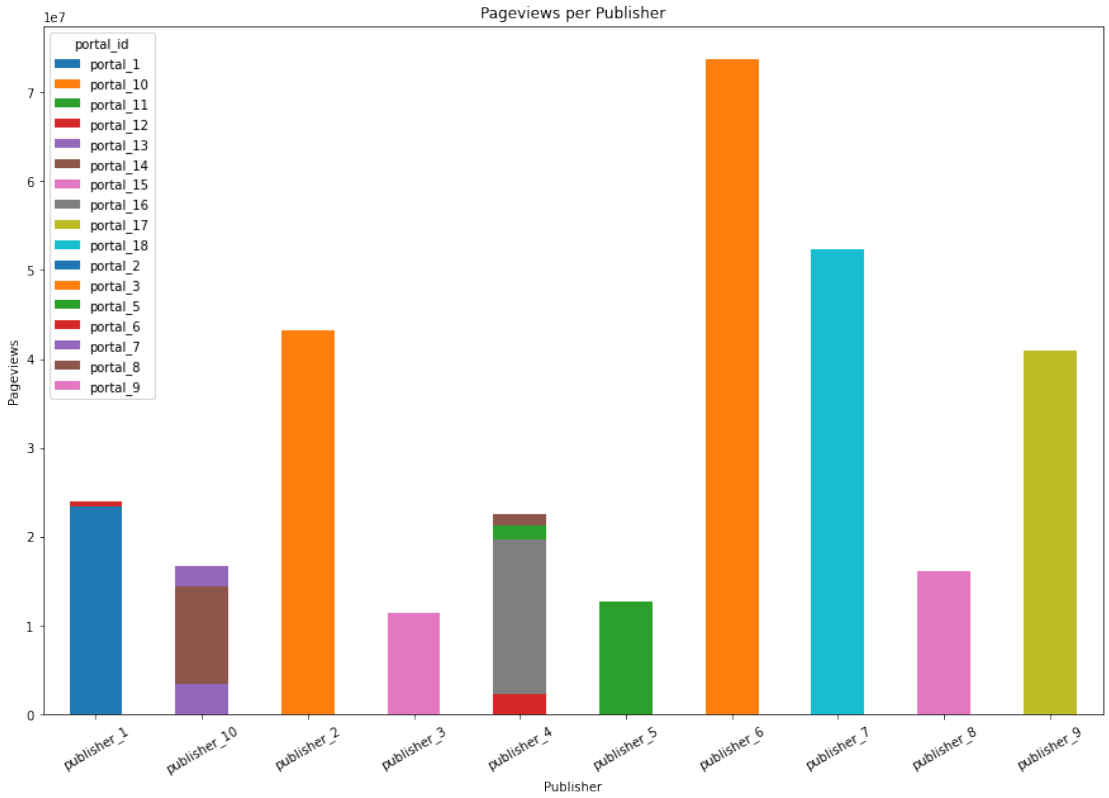


Figure 3.3: Pageviews per publisher with corresponding portals of the publisher.

3.2 Model Architecture Overview

In this section an overviews of the *MARS* model architecture and its implementation is given. The model generates recommendations by creating a user profile representation based on the user's view history and the content representation of the items in the history. Furthermore, the user profile representation is dynamically adapted to the candidate items by generating a representation of the candidate item, which is used for an attention mechanism to weight the different items in the user's history. Through the attention, the proportion of items in the user's history to contribute to the user profile representation, which are similar to the current candidate item, is increased. On this way, the diverse user's interest is reflected by selecting items from the user's history which are more important to reason about the candidate item's usefulness for the user. Figure 3.4 shows an example for the adaptive user representation, where similar items are closer together in the representation space and the distance between the user and the items indicate the interest of the user in the item. A user i liked a list of different kinds of items (j_1, j_2, j_3, j_4). Because of the diversity of the liked items by user i, their corresponding representation vectors (v_{j_1} ,v_{j_2} ,v_{j_3}, v_{j_4}) form two clusters in the space. This results in the fixed user's representation resides between the two clusters. So with a fixed user representation, the *RS* would recommend item s to the user, although item j or k are more close to the items liked by the user. The adaptive user representation on the other hand is able to pay attention to the users diverse interest.[73]

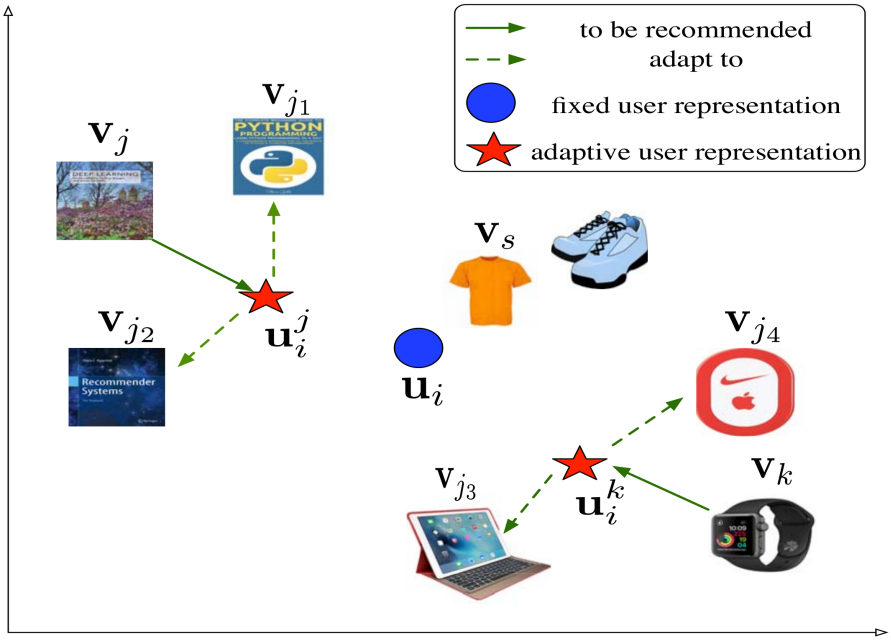


Figure 3.4: "A fixed user representation u_i fails to express user i's diverse interests. Adaptive user representations dynamically adapt to relevant items."[73]

The overall architecture of *MARS* is shown in figure 3.5. The model from the bottom to the top is structure as followed:

Input Y^{n_i} and Y^j represents the documents, which describe the liked item by the user and the item to attend to.

Item Representation Layer $f_{user}(:;\Psi)$ and $f_{item}(:;\Omega)$ are the item representation layers, which generate an item embedding used in the memory component and the context item j. Ψ and Ω represent the learnable parameters of the item modeling layer.

Memory Component & Item Representation C depicts the users memory component, which is the concatenation of the embeddings of all items liked by the user. V_j is the item embedding of the context item which is attended to.

Attention Vector α^{ij} is the attention vector, which weights each item embedding i in the memory component C by $softmax(C^T v_j)$.

Adaptive User Representation The adaptive user representation u_j^i is obtained through the weighted sum of C by $alpha^{ij}$.

Preference Score The final preference score r_{ij} is computed by the inner product of the user and context item embedding $u_i^{jT} v_j$.

The final item recommendation list is obtained by computing the preference score r_{ij} for all candidate items and by sorting the items by their score in descending order.

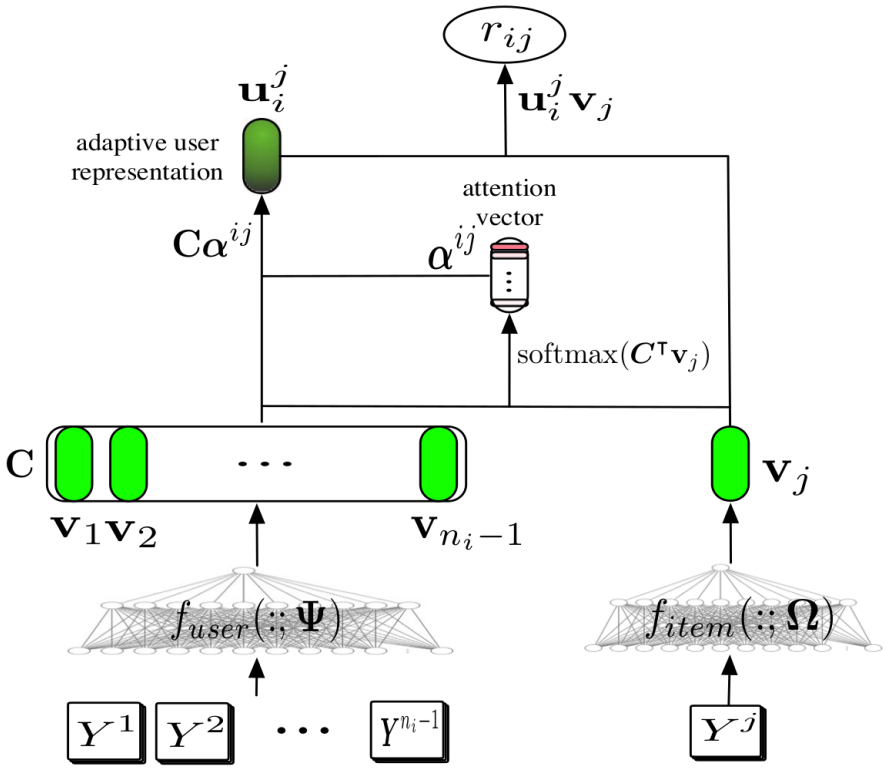


Figure 3.5: Overall MARS architecture. [73]

User Model The users are basically modeled by the summed item representations the users have viewed. The dynamic user representation is achieved by weighting with an attention layer, which attends to the context items.

Item/News Model The item representation is modeled with a CNN according to the architecture outlined in figure 3.6. On the bottom, the models input layer consists of the tokenized words of the input document. On top of that follows a word embedding layer, which transforms the word tokens into embedding vectors. Next, the word embeddings are processed by one convolutional layer. The *CNN* extracts contextual features, where

each kernels height spans the whole word embedding vector, while sliding over the embeddings and is intended to capture one feature. A Max-pooling layer extracts the most important feature value from each kernel. Finally, a dense layer projects the extracted features of all kernels into the final item representation space.

The neural network topology for the items in the user's history and the context items are the same and only differ in the input and model parameters, hence the above description and depiction in figure 3.6 applies to both *CNNs*, $f_{user(...)}$ and $f_{item(...)}$, in the architecture overview of figure 3.5.

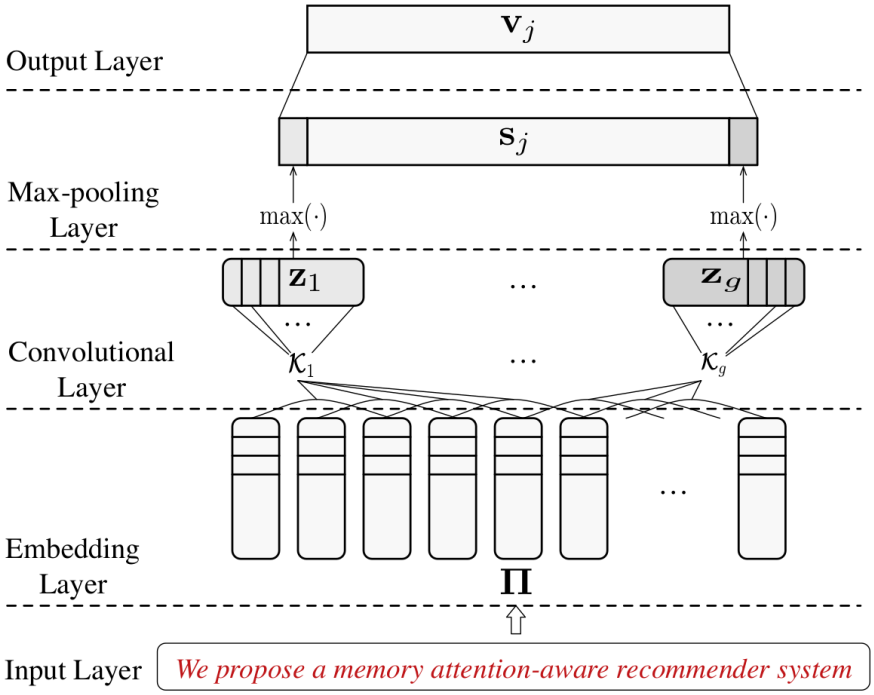


Figure 3.6: MARS CNN architecture used for learning item representations. [73]

Model Training

The training of *MARS* is designed to optimize the model for ranking and the authors took inspiration from [40]. The key aspects they adopted are the process to generate the training data and the loss function.

Training Data The training data generation process is formalized as:

$$\mathcal{D} = \{(i, \mathcal{I}_i^+ \setminus j, j, j') | i \in \mathcal{U} \wedge j \in \mathcal{I}_i^+ \wedge j' \in \mathcal{I}_i^-\} \tag{3.1}$$

Where i, j, and j' are uniformly sampled from $\mathcal{U}$, $\mathcal{I}_i^+$ and $\mathcal{I}_i^-$ and represent the set of users i, the viewed items j by a user and the not viewed items j' of that user. According to [40], this data generation strategy is intended to account for the skew in data, where some few items are overly present in many user's histories and would lead to train excessive on these few items when just iterating the samples. Also there are typically much more not viewed items in contrast to viewed ones. Additionally, [40] states that this sampling strategy leads to a much faster convergence than simply iterating the training data e.g. user wise.

Loss Function To train the model on ranking, the loss function takes pairs of viewed and not viewed items to represent the user's u preference of item j over item j' as shown in equation 3.2. So the model is trained to score each item viewed by the user higher than each not viewed item.

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{(i, \mathcal{I}_i^+ \setminus j, j, j') \in \mathcal{D}} \{ln(\sigma(u_i^{j^T} v_j - u_i^{j'^T} v_{j'})) + \lambda_u u_i^{j^T} u_j + \lambda_u u_i^{j'^T} u_{j'} + \lambda_v v_j^T v_j + \lambda_v v_{j'}^T v_{j'}\} \tag{3.2}$$

Here, u_i^j is the adaptive user representation of user u for item j and v_j is the item representation of j. The counterpart for the not viewed item j' and its corresponding adaptive user representation are $v_{j'}$ and $u_i^{j'}$. The user's i's preference of item j over item j' is mapped into probabilities by the sigmoid function σ. The overall training minimizes the negative log likelihood. λ_u and λ_v are regularization terms.

Model Implementation

Training Model Figure 3.7 shows the schematic representation of the *MARS* implementation with *Tensorflow* and *Keras*. As training input, the model gets an item the user liked, extracted from the view history, the history minus the extracted item and an item the user does not like/has viewed as shown on the bottom and already described in section 3.2. To compute the liked and not liked context item and the corresponding adaptive user representations for the loss computation, two instances of the model with shared weights are applied to the different context items and the same view history. This

is illustrated by the mirrored components on the left and right side of the figure. The turquoise box in the center of the figure represents the *CNN* model, which is used for all user/context item representations, with the only difference being the different model parameters for user and context items. The adaptive user representation is computed by first applying the *User Item Model* to all viewed items of the history through a *Keras Time Distributed Layer*. This layer sees the input as time steps and applies the model stepwise, which yields the *Memory Component*. The *Memory Component* is just an intermediate result, rather than a real model component and represents the *Tensor* holding all user item embeddings. The adaptive user representation is computed by a dot-product attention layer, respectively Luong-style attention, with user item embeddings as keys and values and the context item as query.

A more detailed graph generated by *Tensorboard* from the implemented code can be found in appendix A.1.

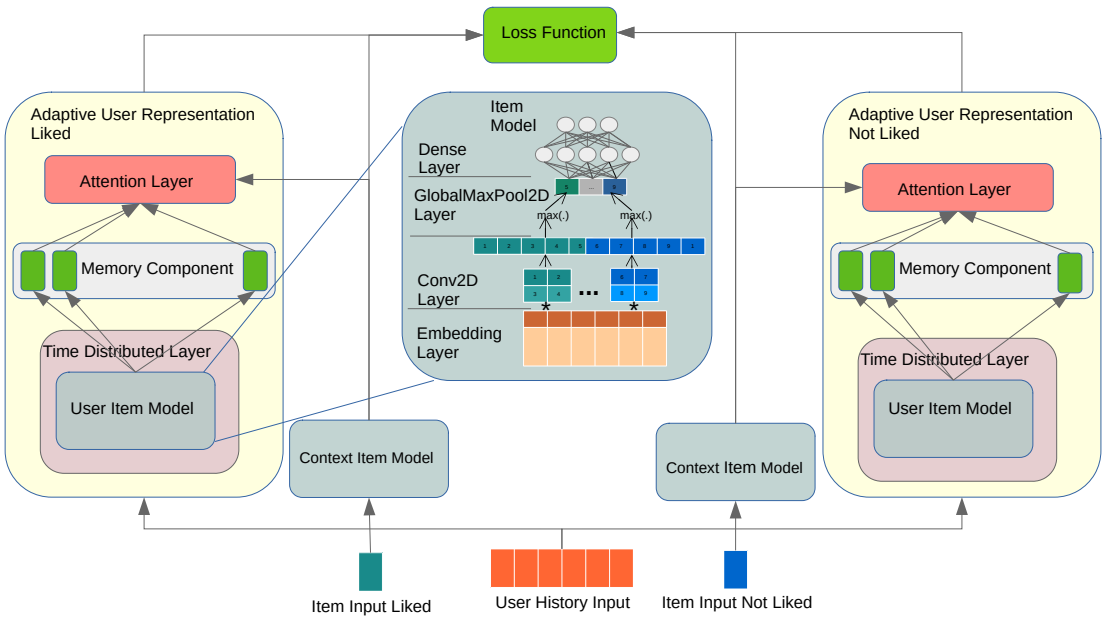


Figure 3.7: MARS implementation with Keras layers

Inference Model Theoretically, the same model from the training can be used for inference ,with additionally just applying a sort over the scores for all candidate items to get a ranked list. But this naive approach will be computationally very expensive and way too slow for real time recommendations as the amount of candidate items is potentially

very large. To avoid recomputations of user and context items embeddings, they are precomputed as shown on the right side in figure 3.8. On The bottom of the inference model, the embedding layers can be seen, which are initialized with the precomputed embeddings, so the model just has to look up the representations for the items. To efficiently compute the scores for all candidates a different *Scores Model* is implemented as seen in the center of the figure. For very compact, expressive, and efficient code, the *einsum* operator in *Tensorflow* is used, which is shown in listing 3.1. The final ranked top k items are computed by a layer, which uses *tf.math.top_k* and *tf.gather* to fully utilize the parallelism and potential usage of GPUs of *Tensorflow*, in contrast to sorting the candidate scores in *Python*.

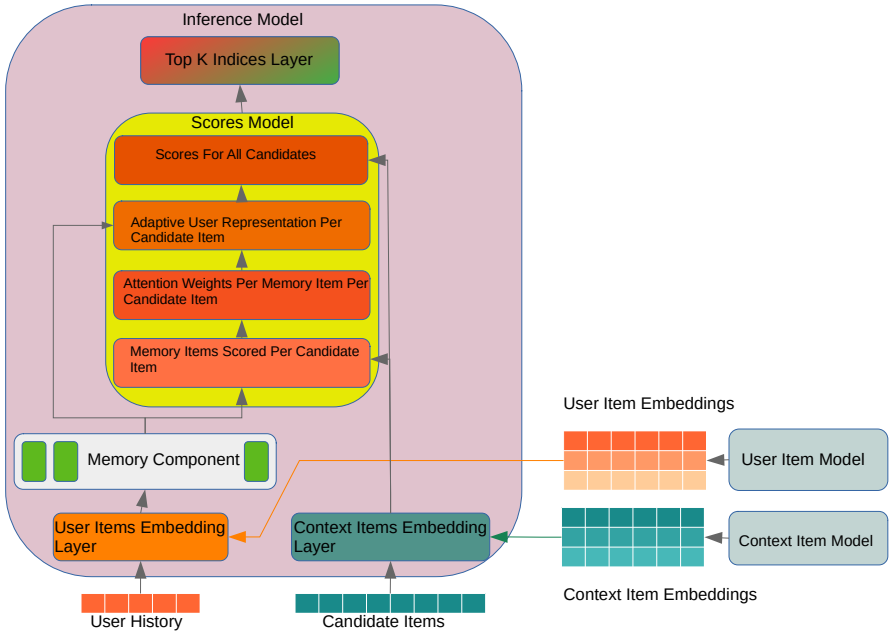


Figure 3.8: MARS inference model

```
import tensorflow as tf
...
# each batch entry corresponds to the memory component of one user
user_memory_batch, items_repr_batch = inputs
# compute the scores for each item with the memory components of each user
scores = tf.einsum("ijk,ilk->ilj", user_memory_batch, items_repr_batch)
# compute the attention for each item to the user memory components for
# all users
attention = tf.nn.softmax(scores, axis=-1)
# multiply the the embedding vector of each user memory component with the
# corresponding item attention and sum the embedding vectors of all memory
```

```
# compoments together to get the deep adaptive user representation
user_repr = tf.einsum("ijk,ilj->ilk", user_memory_batch, attention)
# compute the scores for each item with the deep adaptive user representation
# generated individually for each item
all_scores = tf.einsum("ijk,ijk->ij", user_repr, items_repr_batch)
return all_scores
```

Listing 3.1: Inference model using *tf.einsum()* operator

3.3 Processing Pipeline

The processing pipeline is depicted in figure 3.9. The data processing is structured in the following four main parts with different steps.

Preprocessing The preprocessing steps represent practically the *ETL (Extract, Transform, Load)* process. First, the data has to be decrypted as the *parquet* files are *GPG* encrypted. For a fast decryption, all *parquet* files are decrypted with *GPG* using pythons *Multiprocessing* library for parallelization. The file content for pageviews and articles are streamed to *PyArrow* for further processing. *PyArrow* is used to extract only columns, which are needed for model training and evaluation and filtering rows for null values as well as metadata for specific criteria. Afterwards, the pageviews are grouped into users with their view history. For the *MARS* baseline, the used fields for pageviews are *'user_id', 'article_drive_id', 'fraction_article_read'* and for the articles *'article_drive_id', 'article_full_text'*. To get a sufficient history per user, only users with at least 10 pageviews are selected. Also only pageviews where an article is read at least 20 % are kept, to ensure the user find the information useful. All articles which have no corresponding pageview after filtering the pageviews are excluded. To extract tokens from the articles, the library *Spacy* is used. Also the filtering of stopwords and punctuation is done via *Spacy*. As in the *MARS* paper, words with a frequency less than four are dropped. Finally, the textual representation of the articles is transformed into indices as input to the embedding layers of the model. This conversion is done with *Keras* text processing module. Also, texts which are shorter than the *CNNs* kernel size are zero padded to the width of the kernel.

Training Data Generation After the pre-processing, the training data for the model is processed. The split into training, test and validation data is done before the training begins, all further data processing is done on the fly while training the model using *Tensorflows Data API*. The data sampling and creation of the input tuple, according to 3.2, happens in a python generator. In this step, the length of the user history and document length is cut to the last 50 pageviews, respectively the first 500 word tokens to account for computational and memory constrains. The python generator is consumed by a *TF Dataset*, which handles padding and data prefetch.

Evaluation Data Generation Like the model training data, the data for the model evaluation is created on the fly while feeding the inference model. As the amount of candidate items which have to be filtered is very high, the whole computation occurs within the *Tensorflows Data API* with pure *Tensorflow* operations, which is much faster as python code and automatically parallelized. For the baseline, just the already viewed articles by a user are filtered. A worthwhile addition would be to filter the candidates also for publisher/portal and freshness, which should yield better and faster recommendations for a relative small effort.

Training & Evaluation For training, the model with its parameters are initialized or loaded from a previous checkpoint. The model with its current parameters is saved each epoch and statistics about the training are recorded. At the end of each epoch, the model is evaluated using the inference model. Therefore, the user and context items are precomputed with the fresh trained user and item model and the inference model is initialized with the new user and context item embeddings. After computing the top k candidate rankings for each user, the metrics are calculated.

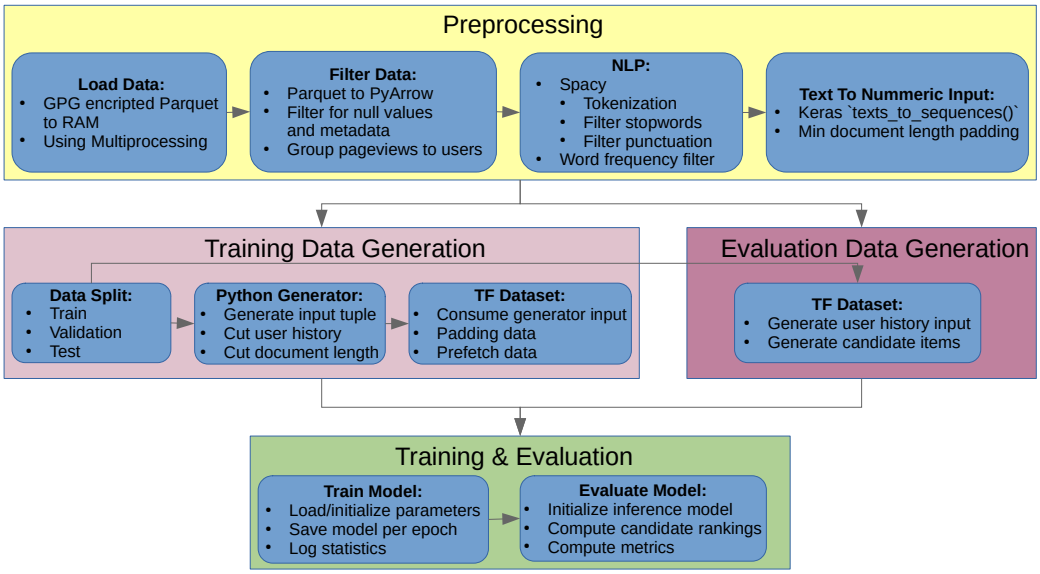


Figure 3.9: (Pre)processing Pipeline

3.4 Adaptations to the News Domain

Adaptions to the model architecture itself are not necessary, because of the already considered modeling of texts. Only the direct integration of pageview and article metadata into the model would require an adjustment. Since the baseline takes advantage of the metadata indirectly in the (pre)processing stage, no modifications on the model architecture were required. The main adaptions concern the preprocessing phase and inference model. Major restructurings are owed to the orders of magnitude more data than used in the *MARS* paper, which required more efficient and parallelized data processing.

Preprocessing Adjustments For the news domain, the possible multiple pageviews of the same article have to be considered in contrast to ratings/likes, which just occur once per item. Further, the loading of data had to be adjusted as the proprietary dataset is stored in *parquet* files protected by *GPG* encryption. Since the python bindings of available libraries to read the *parquet* format do not support the integrated encryption specification of the format yet, the files had to be encrypted with the external *GPG* tool. The integrated encryption would allow to read and decrypt only a subset of the files, but the external encryption made it necessary to load the whole file at once and drop unused

columns afterwards. To accelerate the data loading, each file is decrypted and loaded via its own process from a *ProcessPoolExecutor*. In order not to risk a leak of unencrypted data and reduce complexity of the pipeline, the whole processing happens in memory without any intermediate file caches. The last major change in the processing pipeline was to abandon *Pandas* and just use *PyArrow* operations, because *Pandas* internal data structures have an inefficient memory representation and *PyArrow* operations perform much faster. Missing functionality in *PyArrow*, like *GroupBy* with custom aggregations, are done in python with multiprocessing.

Inference Model Adjustments To compute the metrics for every user with all candidate items as input, the inference model and its input pipeline had to be adjusted. To deal with the massive amount of data, the whole pre-processing for the inference model is moved from *Python/numpy* to *Tensorflow DATA API*. This prevents from converting large model inputs from python to *Tensorflow* data structures and leverages efficient *Tensorflow* operations. Since *Tensorflow* has no similar functionality like *numpy.isin()* and set operations are very slow, this function had be implemented. The usage of a *TF Dataset* also enabled the utilization of all GPUs through a conversion to a *DistributedDataset*. To further improve the performance, the user item embeddings are now precomputed, as the context item embeddings were already before. The precomputed item representations are now used to initialize item embedding layers in the model and the inputs are just item ids instead of embedding vectors. Furthermore, the computation of the top k scores are now part of the model instead of outputting all item scores and sorting them in *Python*. So the sorting of scores is done in *Tensorflow* with support of the GPUs. These steps led from a not working implementation for this amount of data to a ∼60 times throughput increase compared to the first working implementation.

3.5 Toolchain

In this section the used tools are briefly reviewed.

3.5.1 Environment

Docker

Docker[3] is a tool for container virtualization and separates the environment inside a container from its host operating system and other containers running on the host. It was used for easy deployment of the RS on different host systems through a guaranteed consistent environment inside the container, regardless of the host system. Another crucial benefit of *Docker* was to use pre build containers for *Tensorflow* with a pre configured environment for GPU usage.

Jupyter Notebook

Jupyter Notebook[4] is an interactive web-based environment to create notebook documents and execute code on a remote machine. It was used for convenient data exploration, prototyping and execution of some experiments.

3.5.2 Data Format & Processing

Parquet

Parquet[5] is a widely used column based data format for efficient storage with fast file access. The columnar design enables to load only subsets of columns for efficient and fast data analytics. The format allows compression on column level granularity, which enables economical usage of storage with different compression algorithms allowing to tune the files to speed up I/O throughput or saving space. For the news recommendation dataset, it provided a fivefold data compression and fast data loading. Unfortunately, the specified integrated data encryption is not yet available for *Python* libraries, which would bring an additional speedup in data loading and more convenient usage of encryption.

[3] https://www.docker.com/
[4] https://jupyter.org/
[5] https://parquet.apache.org/

Apache Arrow/PyArrow

Apache Arrow[6] is a cross-language framework for columnar in-memory data processing. It defines an efficient columnar in-memory data format, provides basic data processing and a language agnostic shared in-memory object store. The object store provides zero-copy reads on the system between different processes without serialization overhead. It is also integrated into *Parquet* and *Pandas*, which allows to read *Parquet* files directly from *Pandas*. *Arrow* is currently used for loading the data and major preprocessing steps through its *PyArrow Python* bindings.

GPG (GNU Privacy Guard)

GPG[7] is a cryptographic software suite for symmetric and asymmetric cryptography. It was used to initially encrypt the data and encryption in the data pipeline.

Numpy

Numpy[8] is a library, which provides the *ndarray* data structure, a multidimensional array with homogeneous data types and fixed allocated memory, and a large collection of functions operating on these arrays. These data structures provide an efficient memory usage and enable fast computation. Several libraries like *Pandas*, *SciPy* and *scikit-learn* use *ndarrays* as data structure or provide compatibility to it. The same *ndarrays* are used by *Tensorflow* as underlaying tensor representations, which makes *numpy arrays* a good choice as *Tensorflow* input and provides a cheap and easy conversion between *numpy arrays* and *Tensorflow* tensors. This enables also the use of a broad collection of libraries without the need to convert between different data structures in preprocessing. So *numpy arrays* are used at many stages in the processing pipeline and as input to *Tensorflow*.

[6] https://arrow.apache.org/
[7] https://gnupg.org/
[8] https://numpy.org/

Pandas

Pandas[9] is a library for data manipulation and analysis, especially of tabular and time series data. Its data structures are *Datafarmes* and *Series*, which are based on *numpy arrays* and the operations on these allow SQL like queries. *Pandas* was mainly used to analyze the dataset and for preprocessing of the small datasets for validation of the *MARS* implementation. Though the usage of *Pandas* for some preprocessing steps were very convenient, it can not be used for pre-processing of the news dataset as its memory usage was about three times higher than *Arrow Tables* and the processing speed was also not competitive to *Arrow*. Nevertheless, it served well for data exploration and analyses of the news dataset.

Spacy

Spacy[10] is an *NLP* library, which offers tokenization, part-of-speech tagging, dependency parsing, text categorization, named entity recognition and more. It was used in the the data pipeline to tokenize the news articles and filter out stop words and punctuation. For this, the ML model *de_core_news_sm*[11] in version *3.3* was utilized, which is trained on German news data.

3.5.3 ML Framework

Keras

Essentially, *Keras*[12] is a high level API for *Tensorflow*. It provides high level abstractions to create ML models on top of *Tensorflow*, without using low level operations and thus makes ML model creation faster. Once supporting several ML frameworks as a backend, it is now an integrated part of *Tensorflow*, which is the sole backend available. As *Keras* is the official high level API for *TF*, it provides an excellent integration. The easy creation of models and the possibility to seamlessly interweave high level *Keras* and low level *Tensorflow* operations was the reason to decide for *Keras*. So *Keras* was used to create the implemented model and as it provides some text processing utilities, also for

[9] https://pandas.pydata.org/
[10] https://spacy.io/
[11] https://spacy.io/models/de
[12] https://keras.io/

some preprocessing steps. The *tokenizer* from *Keras* is used to create a vocabulary index and to transform the article texts into integer tokens, so each article is represented as a vector of integers. Additionally, the *tokenizer* was used to filter the words by frequency to reduce the vocabulary.

TensorFlow

Tensorflow (TF) [13] is the ML framework doing the actual computation, automatic back-propagation and distributed execution on clusters and different devices like CPUs, GPUs or TPUs. TF has two execution modes, an *eager* called mode with a *Define-by-Run* execution like *PyTorch*, which enables a step by step execution of operations with easy debugging and a static execution mode, where operations define a computational graph in the form of a directed acyclic graph (DAG), which enables advanced automatic optimizations and better performance. TF was chosen for its high performance, scalability, broad community support and wide adoption.

3.6 Discussion

Considering the dataset analysis and computational requirements through the high dimension of the data, a limitation of the view history and article length to a maximum of 50 pageviews and 500 words is chosen. For the history, the last 50 pageviews are used and the first 500 words for articles. This is well above of the average pageviews per user of 5,58 and also above the average article length of 398,35 and should not limit the recommendation performance too much. The article length only affects the full text and to a small extend the teaser text. The header text length is always below the 500 limit. To get a sufficiently long user history for effective training, just users with at least 10 pageviews are considered. This is also necessary since the experiments are conducted with a training, validation and test data split like the *MARS* paper of 30 %, 35 % and 35 % and the split is performed on the history of the user instead of splitting between users. Furthermore, to just consider pageviews of articles the users liked, the minimum fraction a user read is set to 20 %, which is a bit below the average of 24 %. This was chosen, because the user may have taken enough information for their need with the first few lines without reading the whole article but pageviews where the user has noticed very

[13] https://www.tensorflow.org/

quickly the article is not interesting for them should be ruled out. With this constellation the resulting training data and number of model parameters are summarized in table 3.4.

Content	Vocabulary Size	# Users	# Items	# Ratings	# Parameters
article_header	34 665	722 241	359 332	31 713 946	18 441 648
article_teaser	72 093	722 241	359 332	31 713 946	38 274 648
article_full_text	442 282	722 241	359 332	31 713 946	216 153 048

Table 3.4: Training data after filtering for article header, teaser and full text.

Hyperparameter	Value
Embedding Size	300
CNN Kernel Width	5
Batch Size	512
Latent Vector Dimension	64
# CNN Filters	64
Learning Rate	0,001
λ_u	0,002
λ_v	0,002

Table 3.5: Hyperparameter for training.

The different parameter sizes come from the different vocabulary sizes resulting in different amount of word embeddings. The amount of parameters stemming from the item representation is actually relatively small compared to the parameters, due to word embeddings. The training will be conducted with the same hyperparameters for all runs, which are mostly the same as in the *MARS* paper, see table 3.5. Only the *Latent Vector Dimension* with 64 is between the 50 and 70 in the *MARS* paper, as a size with a multiple of 8 is beneficial for optimal usage of GPU resources.

In this chapter further questions are answered:

Q1: The final part for the question how to establish a baseline for this dataset was to explain the model architecture, describe the implementation, including training procedure and processing pipeline, and the necessary adjustments for the news domain, which compromise mainly the optimizations to account for the very large dataset. As the *MARS* model includes already a text modeling part, no changes on the model architecture were needed.

Q2: In section 3.1 the dataset was analyzed and data fields for the experiments were selected. The final used fields are *user_id*, *article_drive_id*, *fraction_article_read*, *article_header*, *article_teaser* and *article_full_text*. Other very useful identified fields are *publisher_id*, *portal_id*, *geo* record, *session_id*, *session_referer_medium*, *session_referer_source*, *is_plus_article*, and *article_dpa_id*. These may be used

in further experiments to examine if they can contribute to enhance the recommendation performance by including them directly in the model input or using them for candidate selection.

Q3: The challenges in utilizing a real world dataset for an RS was actually the sheer amount of data to be processed. The received data was actually well structured with clean text, without the need to extensively filter for special characters or missing content. Where the clean text is on the one hand due to the news domain and on the other hand due to receiving the data from the creators instead of scraping it from the web.

Q4: Challenges in the adaptation of a general RS model to the news recommendation domain, in the context of the concrete model and implementation, was to implement the model in a way to efficiently handle very long texts in comparison to movie synopsis or product descriptions. In general, the high dimensionality of the data, with $[\textit{batch size} \times \textit{history length} \times \textit{document length} \times \textit{embedding size}]$ was challenging for performance reasons and available GPU memory.

4 Evaluation

4.1 Metrics

The metrics to evaluate the baseline are *recall@N* and *Mean Average Precision(MAP)* like in the *MARS* paper, shown in equation 4.1 and 4.2. Recall is used to measure how good the model can retrieve all relevant documents, respectively how much of the actual viewed articles of the users are recommended by the model. MAP is used to measure how well the model can rank the retrieved documents, respectively if relevant articles ranked higher than not relevant articles. These metrics are suitable to the news recommendation scenario, because they evaluate how well the RS generates the typical used top n lists. It is also important to note, that these metrics do not measure the ordering between relevant articles. Deviating from the usual method to make the validation and test data split per user, the pageviews per user are split into training, validation and test data according to *MARS*[73], to make the results comparable with the original experiments. So one have to keep in mind, that it is measured how good the model generalizes to new views of a user and not how good it generalizes to new users. For scenarios where the model is continuously and rapidly trained on new data, this makes sense as the majority of recommendations will be provided to known users. If the model will not be updated in a short period of time or many new users use the service during model update, it would make sense to take the measures on a traditional per user data split to account for unknown users. For resource reasons, the tests are just run one time. For more expressive results and to measure the standard deviation, it would be appropriate to run each test at least three, better five to ten, times.

$$recall@N = \frac{\#\text{ items the user likes among the top N}}{\text{total number of items the user likes}} \tag{4.1}$$

$$
\begin{aligned}
MAP &= \frac{1}{|U|} \sum_{i \in U} AveP(i) \\
AveP(i) &= \frac{1}{|K|} \sum_{k=1}^{K'} P_i(k) rel_i(k) \\
P_i(k) &= \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} \\
rel_i(k) &= \begin{cases} 1, & \text{if } k \text{ in } \{\text{relevant documents}\}. \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
\tag{4.2}
$$

Where $AveP(i)$ describes the average precision for user i. $P_i(k)$ denotes the precision for user i for the top k articles recommended to user i and $rel_i(k)$ the relevance of the k_{th} document for user i, which leads to irrelevant documents being ignored.

4.2 Results

In this section, the results of the three runs for *article header*, *article teaser* and *article full text* as content data are shown, and as comparison the results from the previous work [29] for validating the model implementation with the original used datasets. For every training run the model was trained on 9 514 184 samples which are divided into 10 epochs. Figure 4.1 shows the *recall@50* metric for all runs and 4.2 the *recall@50* for the model validation experiments. Figure 4.3 shows the *MAP@500* metric for all runs and 4.4 the model validation experiments for the *MAP@500* metric. The green graph in figure 4.1 and 4.3 represents the *header* content, the yellow graph the *teaser* content and the red one the *full text* content. In figure 4.5 the losses of all runs are shown with grey for the *header*, dark blue for *teaser* and light blue for *full text*. Note that in figure 4.1 and 4.3, the zero epoch represents the measure for the untrained model, with initial weights, to show the difference between the untrained model with model after the first training epoch. The loss for the zero epoch in figure 4.5 represents the value after the first epoch of training. So to relate the figures one has to offset the loss epoch with 1.

In figure 4.1 it can be seen that the runs with teaser and full text content clearly outperform the run with the header text at the *recall@50* metric. The run with header text reached its peak at epoch 10 with 0,0268, teaser text at epoch 5 with 0,0387 and full

text at epoch 7 with 0,0362. The teaser text run reached an about 6,9 % higher score than the full text run and a 44,4 % higher score than with header text. All three runs have in common that after the first epoch they reached a score, which is relatively close to its peak performance. For comparison, the runs for the original datasets in figure 4.2 show a pretty similar behavior with a steep ascent in the first epochs, which comes very close to its peak performance. The big difference is the extremely higher score for the *Yahoo! Movies* dataset with 0,4449 (original paper 0,3230), which is about 11,5 times higher than for the news recommendation dataset. The Runs for *Amazon Video Game* and *Amazon Movies and TV* with 0,038 44 (original paper 0,1337) and 0,0227 (original paper 0,1196) come very close to the news dataset with teaser text, when not considering the original papers numbers.

Figure 4.3 shows that the runs with teaser and full texts also outperforms the header text runs at the *MAP@500* metric. The peak performance for header texts is reached at epoch 10 with 0,0043, teaser texts at epoch 7 with 0,0126 and full texts at epoch 2 with 0,0106. The teaser texts run reached an about 18,86 % higher score than the full text run and a 193 % higher score than with header text. Similar to the *recall@50* metric figure, all three runs reach a score close to its peak performance relative early. In comparison, the original papers datasets show also a very steep ascent, closely to its peak performance, in the first epochs. Again the difference is high with 0,136 (original paper 0,1692) for *Yahoo! Movies*, 0,005 265 5 (original paper 0,0934) for *Amazon Video Game* and 0,002 837 2 (original paper 0,0895) for *Amazon Movies and TV*. This time the score for the *Yahoo! Movies* dataset is 10,8 higher than for the news dataset but the teaser text run reaches 2,4 times and 4,4 times higher scores than for the *Amazon Video Game* and *Amazon Movies and TV* dataset.

The loss in figure 4.5 shows that on all runs the model learns continuously without fluctuations and a typical curve with an initial steep descent, that becomes increasingly flat. If one relate the loss with the *recall@50* metric, it indicates, that training on more epochs would probably increase the recall and the model do not overfit. If one relate the loss with the *MAP@500* metric, it indicates that the model, at least for the full text run and possibly for teaser texts, is overfitting on the data and does generalize worse with more training just after a few epochs.

Table 4.1 shows the highest scores for different top k *recall@* and *MAP@* and lowest losses for the training runs with *article header*, *article teaser*, and *article full text* content data. The corresponding epoch, where the highest score/lowest loss were reached, are in

parenthesis with plus 1 for loss epoch to account for the mentioned offset. The highest score/loss in a row is marked bold. It can be seen, that the model with teaser text reaches the best scores, regardless of the chosen top k. Interestingly, the advantage of the model trained on teaser text is especially high at the low top k of 5, 10 and 20 which are the most important positions, as these are the most noticed positions in top recommendation lists on websites.

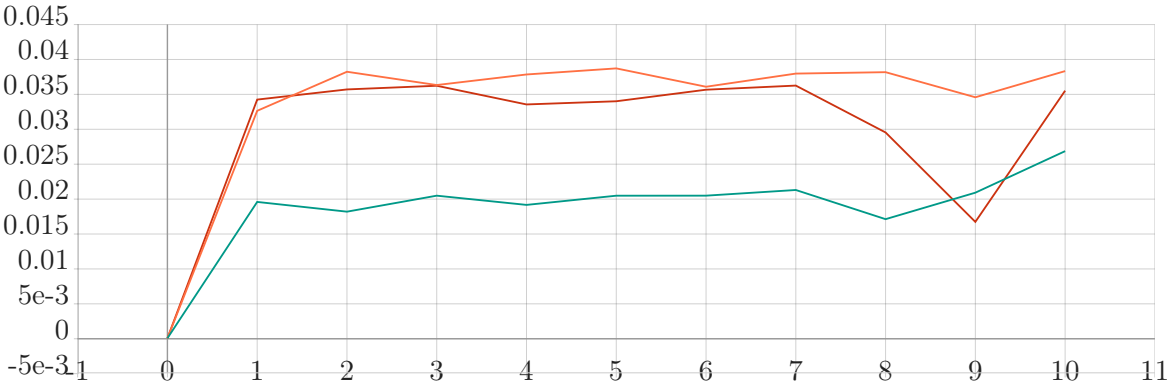


Figure 4.1: *Tensorboard* scalar graph of MARS Recall@50 metric for content header, teaser, and full text.

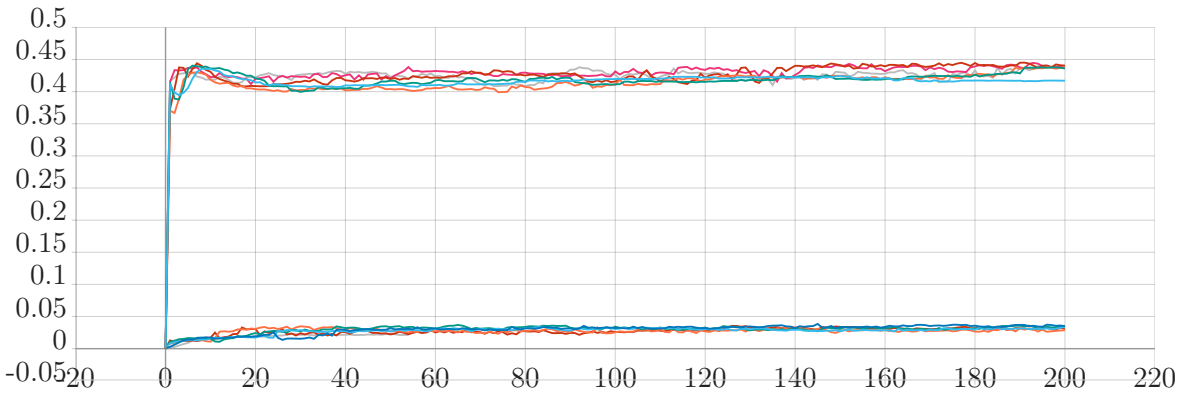


Figure 4.2: *Tensorboard* scalar graph of MARS Recall@50 metric for the original papers tested datasets.[29]

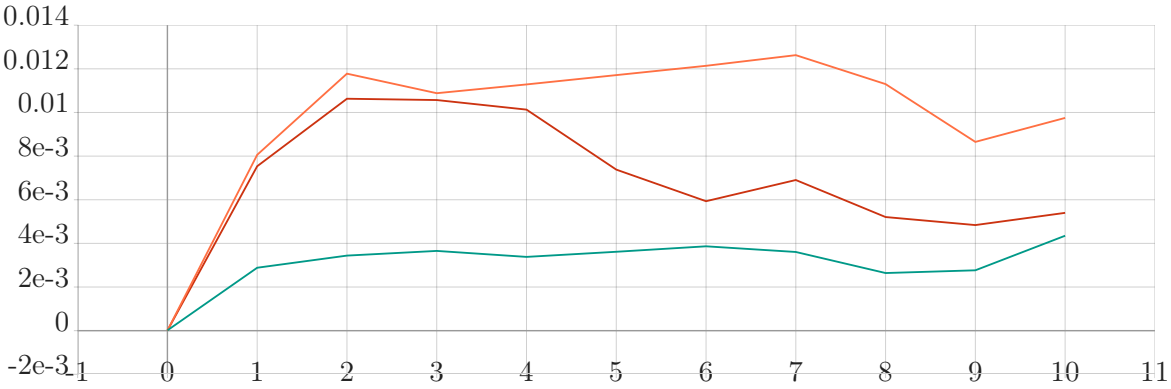


Figure 4.3: *Tensorboard* scalar graph of MARS MAP@500 metric for content header, teaser, and full text.

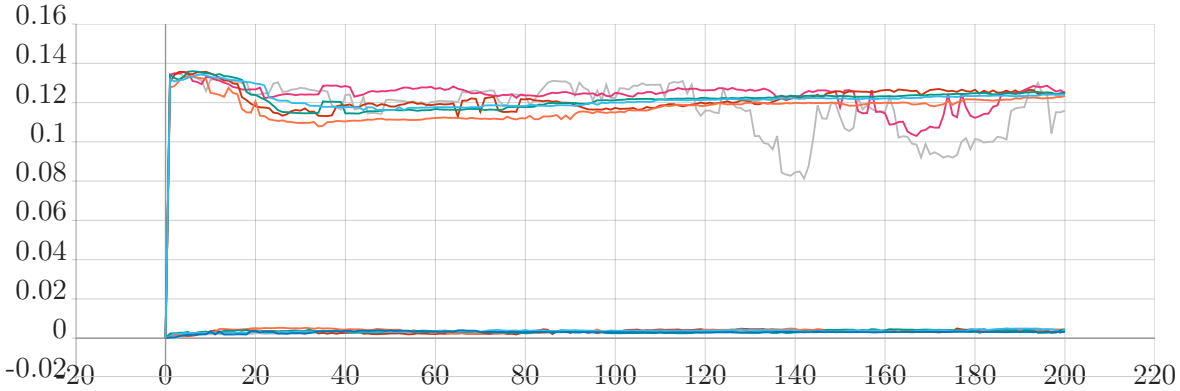


Figure 4.4: *Tensorboard* scalar graph of MARS MAP@500 metric for the original papers tested datasets.[29]

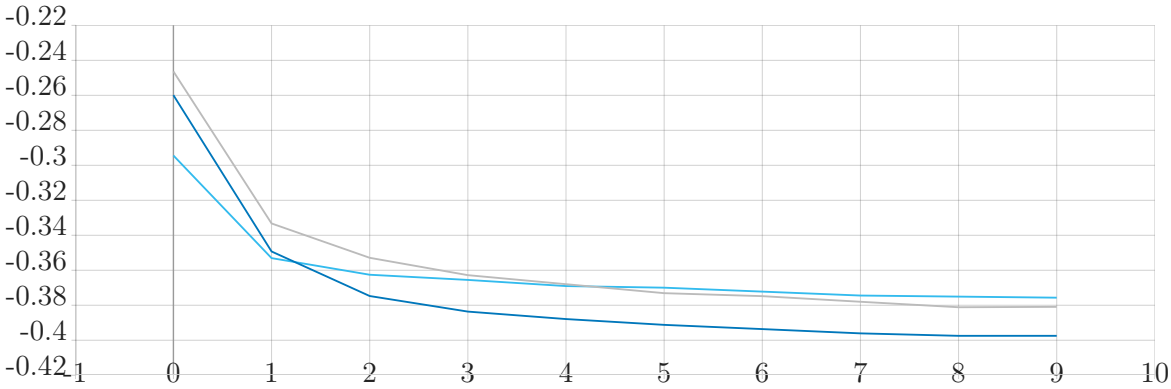


Figure 4.5: *Tensorboard* scalar graph of epoch loss metric for content header, teaser, and full text.

Metric	*Article Header*	*Article Teaser*	*Article Full Text*
recall@5	0,0042 (3)	**0,0150** (5)	0,0114 (3)
recall@10	0,0116 (10)	**0,0195** (7)	0,0188 (5)
recall@20	0,0160 (10)	**0,0279** (7)	0,0248 (1)
recall@50	0,0268 (10)	**0,0387** (5)	0,0362 (7)
map@5	0,0020 (10)	**0,0097** (7)	0,0079 (3)
map@10	0,0028 (10)	**0,0104** (7)	0,0085 (3)
map@20	0,0031 (10)	**0,0111** (7)	0,0092 (2)
map@50	0,0035 (10)	**0,0115** (7)	0,0096 (2)
map@500	0,0043 (10)	**0,0126** (7)	0,0106 (2)
loss	−0,3811 (9)	**−0,3974** (9)	−0,3756 (10)

Table 4.1: Recall and MAP metrics for different k for the three training runs for header, teaser and full text.

4.3 Discussion

The results of the experiments have shown that the teaser text is very well suited to model the article for news recommendation, and superior to the headline and the articles full text, which answers question **Q7** whether the full text performs better for recommendations as the headline and teaser text. The lower performance of the headlines could possibly be attributed to two properties. First, with average word length of 6,35 it is very short and can not contain as much information about the article content as the full and teaser text. Second, headlines are often lurid and contain less meaningful statements about the actual article content as initially assumed, see appendix A.3 for an example article from the corpus. The superior performance of the teaser texts could be due to that they contain very condensed information and very much important information from the full article content. Also, teasers are meant to support one in the decision to read the article or not, which is exactly the information which is useful for the RS to make a recommendation. The performance of the full text lies between the headline and teaser text, which may be due to it holding the full information of the article but it is much more difficult for the model to capture the meaning of the text. In compression to the teaser, the full text is not already filtered for the most important information and due to the much longer text, the model has to capture dependencies for a much longer range. Another explanation why the full text is not as effective for training as the teaser text may be the different vocabulary and thus parameter sizes, already shown in table 3.4. The large parameter size makes the model harder to optimize but does not necessarily

help to better understand the content of an article. Many words, which occur rarely make the dataset hard to learn, because the embeddings for infrequent words, which occur only a few times in a dataset, are only optimized on a few examples, which makes them not suitable for representing these words. Also, the word structure from infrequent words is overlooked, so the information from frequent words can not be used by the model for infrequent words. This is especially impactful for morphologically-rich languages like German with a high combinatorial count of word forms, used in this dataset [26, p. 76]. In comparison to the datasets used in the *MARS* paper, the results on the news dataset are significantly worse. Taking the different densities of the datasets into account, which is 354,42 times lower for the news than the *Yahoo! Movies* dataset, puts the results into another light. So the news dataset is much more difficult for the model to learn.

In the following, some improvements are discussed, which are independent from the model architecture. A way to overcome problems with a large vocabulary, without changing the model architecture, is to use sub-word tokenization like *BPE* to reduce vocab size without loosing information in comparison to limit the vocabulary to a fixed size or filter out infrequent words. So using *BPE* to reduce vocab size and simultaneously increase other model parameters would potentially improve results by giving the model more capacity to represent the content and interactions of words and do not distract it by optimizing many embedding parameters. Also to improve results, the split between training, validation and test set could be changed to raise the amount of training data. Smaller validation and test sets should not reduce the meaningfulness of the results, given the size of the dataset. Additionally the use of normal sample splits between users and not pageviews per user would be helpful to increase the possible training size and makes training with as low as two pageviews possible and would make the model train to recommend for completely new users. Also a mixed split between samples of users with the full history and hold out pageviews would help the measure generalization for new and known users. Additionally, the candidate item selection can be improved. In this experiment just already viewed articles are filtered out. One has to realize that the dataset is constituted of different sub datasets as it is the aggregated data of several portals/publishers. So a candidate item selection with just selecting articles from the portal the user uses, can greatly improve results. The missing portal/publisher filter may explain the drop in the *MAP@500* metric while the *recall@50* metric may profit from more training. Different portals may have similar articles, while training for more epochs results in catching more relevant articles, the ranking of the articles is more difficult since actually relevant articles are ranked high because they have a very similar representation to the actual viewed article but

do not appear in the users history because they are from another portal. So the model may not really overfit but gets better to generate similar embeddings for similar articles and articles which are very similar to the actually viewed ones but originate from other portals are ranked high. Further, filtering for articles, which are much older than the last viewed item in history and for articles in the future by taking the date of the next pageview in the history which are held out, can reduce the candidate items significantly and ease the final article selection. Lastly, no hyperparameter tuning were conducted due to to little resources.

5 Conclusion & Outlook

5.1 Conclusion

This thesis gave a broad overview over current *LM* and *RS* with a special focus on the news recommendation domain and introduced a proprietary dataset with news articles and pageviews from several publishers and portals. After the analysis of current *RS*, the *MARS* model was chosen to create a baseline for this dataset and to answer further research questions. The analysis could already answer some of the research questions. Question **Q1** was partly addressed by analyzing the requirements for news *RS* and choosing the *MARS* model as baseline. **Q4** was partly answered by identifying the news modeling as a critical part and special characteristics of news *RS* like short lifecycles of news with relatively few user interactions per article compared to e-commerce items. For answering **Q5**, techniques for incorporating metadata from the different reviewed *RS* models could be referred to, like the concatenation of embedded categorical features and dense features into one representational item embedding, which gets further processed by deeper model layers. Metadata could also be used indirectly either for pre-processing the data or using it as labels for auxiliary tasks. The next chapter dealt with the experimental design, where first the news dataset was analyzed and addressed **Q2** by selecting data fields like the head, teaser and full text or the fraction a user read of an article. Useful fields for further experiments are for example the record with geo data or session id. The section describing the model architecture and adaptations to the news domain addressed the technical part of **Q1**. The model was adapted to the news domain, including implementation of the model and processing pipeline from scratch due to a lack of an open source implementation. The described challenges in the use of a real world dataset was mainly about dealing with the huge amount of data, which answered **Q3**. As the selected *MARS* model already was suitable to operate on text data, the main challenge in adapting the model was to implement the model to efficiently handle long article texts and the high dimensional data, which addressed **Q4**. Surprisingly, the

experiments of the evaluation showed that the model reached the best results with the article teaser text as content to model the news articles, which could probably be explained by the fact that teaser texts are very condensed and contain the most important information about the article, which may help the model to grasp the article's content. Even when the full article text could be exploited by stronger *LM*, the teaser text may be nevertheless good enough for an *RS*, especially when considering the computational requirements as the full text needed about 6 times more training time. The header text worked worst, which may not surprise as it is very short and headlines are often lurid and not necessarily reflect the article's content sufficiently. So the evaluation was able to answer **Q7**. Question **Q6** could not be answered in this thesis, due to short resources to extensively run experiments with different metadata and is subject to further work. In conclusion, the analysis of the dataset has shown great potential for personalized news recommendations, and the usage of advanced techniques from the reviewed *RS* and *LM* could improve the user satisfaction and engagement, which will result in a raised number of pageviews and a higher fraction of an article the user reads.

5.2 Outlook

5.2.1 Transferability & Limitations of the approach

The selected *MARS* model can be used in any domain where the items can be represented by an embedding of their content, like text, picture or sound. The effectiveness is mainly affected by how strong the item representation model is and by the dataset domain. As shown by the evaluation, the performance of an *RS* model is highly dependent on the dataset, especially the domain of the data. This is highlighted by the huge difference between the *Yahoo! Movies* and the news dataset. Even the *Amazon* datasets used in the original *MARS* paper did not work as well in the model validation experiments, probably through different pre-processing methods. It has to be noted, that the *MARS* model was developed on datasets, which are very different from news data. The recommendation of news is a much harder task than e.g. e-commerce item recommendation, since the relevance of news articles has a shorter time period and they have fewer user interactions per item, which makes it more difficult to relate similar users. Many factors have to be considered for news recommendation as shown in section 2.2.2, for instance the news representation model plays a key role. Another limitation is the lack of sequence data usage. If the next preferred item of the user is highly influenced by the order, the user has

accessed previous items, like streaming music recommendation, the model is not able to distinguish different permutations of the user's history order. Domains where it depends more on the entirety of the viewed items like recommending relevant research papers for a selection of paper of a topic, the model may perform relatively well. Additionally, the expressiveness of offline experiments is limited in relation to online A/B test in a production deployment. E.g. the model may recommend articles, which indeed interests the user but will be counted as false recommendation, because the item is not in the user's history.

5.2.2 Future Directions & Improvements

Furthermore, the following list shows possible improvements and future research directions:

- The usage of sub word tokenization like *BPE* may improve the results and shrink the number of parameters for large vocabularies.
- Replacing the attention mechanism with a sequential model like *BERT4Rec* and extend it by the training method of the *ELECTRA LM* could massively improve the performance and especially the sample-efficient training.
- Also using *LM* for article representation as shown by [59], will be key to better understand the article content and improve recommendations.
- Methods for simultaneously learning from different data modalities like [1], could improve the article representation by also considering pictures or videos in addition to the text content.
- *DRL* models are a very promising direction for training a model online and will be essential to actively explore the user's interest.
- Explainable recommendations are also a notable research direction, where a comprehensive overview is given by [69].

Acknowledgment

I would like to thank Prof. Dr. Kai von Luck for supervising and supporting this thesis, providing valuable suggestions and his relentless personal commitment. Furthermore, I would like to thank Dr. Christoph Mayer from *SCHICKLER Unternehmensberatung GmbH* for providing the dataset. Finally, I would like to thank all current and past members of the Machine Learning Group (ML-AG) of the HAW Hamburg for great discussions and the team of the Creative Space for Technical Innovations (CSTI)[1] for providing the infrastructure.

[1] https://csti.haw-hamburg.de/

Bibliography

[1] Baevski, Alexei ; Hsu, Wei-Ning ; Xu, Qiantong ; Babu, Arun ; Gu, Jiatao ; Auli, Michael: data2vec: A General Framework for Self-supervised Learning in Speech, Vision and Language. In: *arXiv:2202.03555 [cs]* (2022), Februar. – URL http://arxiv.org/abs/2202.03555. – Zugriffsdatum: 2022-03-15. – arXiv: 2202.03555

[2] Bao, Hangbo ; Dong, Li ; Wei, Furu ; Wang, Wenhui ; Yang, Nan ; Liu, Xiaodong ; Wang, Yu ; Piao, Songhao ; Gao, Jianfeng ; Zhou, Ming ; Hon, Hsiao-Wuen: UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training. In: *arXiv:2002.12804 [cs]* (2020), Februar. – URL http://arxiv.org/abs/2002.12804. – Zugriffsdatum: 2022-04-11. – arXiv: 2002.12804

[3] Bianchi, Federico ; Yu, Bingqing ; Tagliabue, Jacopo: BERT Goes Shopping: Comparing Distributional Models for Product Representations. In: *arXiv:2012.09807 [cs]* (2021), Juni. – URL http://arxiv.org/abs/2012.09807. – Zugriffsdatum: 2021-11-05. – arXiv: 2012.09807

[4] Bordes, Antoine ; Usunier, Nicolas ; Garcia-Duran, Alberto ; Weston, Jason ; Yakhnenko, Oksana: Translating Embeddings for Modeling Multi-relational Data. (2013), S. 9

[5] Brown, Tom B. ; Mann, Benjamin ; Ryder, Nick ; Subbiah, Melanie ; Kaplan, Jared ; Dhariwal, Prafulla ; Neelakantan, Arvind ; Shyam, Pranav ; Sastry, Girish ; Askell, Amanda ; Agarwal, Sandhini ; Herbert-Voss, Ariel ; Krueger, Gretchen ; Henighan, Tom ; Child, Rewon ; Ramesh, Aditya ; Ziegler, Daniel M. ; Wu, Jeffrey ; Winter, Clemens ; Hesse, Christopher ; Chen, Mark ; Sigler, Eric ; Litwin, Mateusz ; Gray, Scott ; Chess, Benjamin ; Clark, Jack ; Berner, Christopher ; McCandlish, Sam ; Radford, Alec ; Sutskever, Ilya ; Amodei, Dario: Language Models are Few-Shot Learners. In:

arXiv:2005.14165 [cs] (2020), Juli. – URL http://arxiv.org/abs/2005.14165. – Zugriffsdatum: 2022-03-05. – arXiv: 2005.14165

[6] Cai, Guohao ; Li, Xiaoguang ; Dai, Quanyu ; Wang, Gang ; Dong, Zhenhua ; Zhang, Chaoliang ; He, Xiuqiang ; Shang, Lifeng: Dual Sequence Transformer for Query-based Interactive Recommendation. In: *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*, Juni 2021, S. 139–144. – ISSN: 2375-0324

[7] Chen, Qiwei ; Zhao, Huan ; Li, Wei ; Huang, Pipei ; Ou, Wenwu: Behavior Sequence Transformer for E-commerce Recommendation in Alibaba. In: *arXiv:1905.06874 [cs]* (2019), Mai. – URL http://arxiv.org/abs/1905.06874. – Zugriffsdatum: 2020-07-29. – arXiv: 1905.06874

[8] Cheng, Heng-Tze ; Koc, Levent ; Harmsen, Jeremiah ; Shaked, Tal ; Chandra, Tushar ; Aradhye, Hrishi ; Anderson, Glen ; Corrado, Greg ; Chai, Wei ; Ispir, Mustafa ; Anil, Rohan ; Haque, Zakaria ; Hong, Lichan ; Jain, Vihan ; Liu, Xiaobing ; Shah, Hemal: Wide & Deep Learning for Recommender Systems. In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. New York, NY, USA : ACM, 2016 (DLRS 2016), S. 7–10. – URL http://doi.acm.org/10.1145/2988450.2988454. – Zugriffsdatum: 2018-05-30. – ISBN 978-1-4503-4795-2

[9] Clark, Kevin ; Luong, Minh-Thang ; Le, Quoc V. ; Manning, Christopher D.: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In: *arXiv:2003.10555 [cs]* (2020), März. – URL http://arxiv.org/abs/2003.10555. – Zugriffsdatum: 2022-03-19. – arXiv: 2003.10555

[10] Dai, Zihang ; Yang, Zhilin ; Yang, Yiming ; Carbonell, Jaime ; Le, Quoc V. ; Salakhutdinov, Ruslan: Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In: *arXiv:1901.02860 [cs, stat]* (2019), Juni. – URL http://arxiv.org/abs/1901.02860. – Zugriffsdatum: 2022-03-23. – arXiv: 1901.02860

[11] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *arXiv:1810.04805 [cs]* (2019), Mai. – URL http://arxiv.org/abs/1810.04805. – Zugriffsdatum: 2020-08-07. – arXiv: 1810.04805

[12] Feng, Chong ; Khan, Muzammil ; Rahman, Arif U. ; Ahmad, Arshad: News Recommendation Systems - Accomplishments, Challenges amp; Future Directions. In: *IEEE Access* 8 (2020), S. 16702–16725. – Conference Name: IEEE Access. – ISSN 2169-3536

[13] Ferragina, Paolo ; Scaiella, Ugo: TAGME: On-the-fly Annotation of Short Text Fragments (by Wikipedia Entities). (2010), S. 4

[14] Goodman, Joshua: A Bit of Progress in Language Modeling. In: *arXiv:cs/0108005* (2001), August. – URL http://arxiv.org/abs/cs/0108005. – Zugriffsdatum: 2022-03-09. – arXiv: cs/0108005

[15] He, Pengcheng ; Liu, Xiaodong ; Gao, Jianfeng ; Chen, Weizhu: DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In: *arXiv:2006.03654 [cs]* (2021), Oktober. – URL http://arxiv.org/abs/2006.03654. – Zugriffsdatum: 2022-03-21. – arXiv: 2006.03654

[16] He, Xiangnan ; Liao, Lizi ; Zhang, Hanwang ; Nie, Liqiang ; Hu, Xia ; Chua, Tat-Seng: Neural Collaborative Filtering. In: *Proceedings of the 26th International Conference on World Wide Web*. Republic and Canton of Geneva, Switzerland : International World Wide Web Conferences Steering Committee, 2017 (WWW '17), S. 173–182. – URL https://doi.org/10.1145/3038912.3052569. – Zugriffsdatum: 2018-06-04. – ISBN 978-1-4503-4913-0

[17] Hidasi, Balázs ; Karatzoglou, Alexandros: Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018), Oktober, S. 843–852. – URL http://arxiv.org/abs/1706.03847. – Zugriffsdatum: 2022-04-02. – arXiv: 1706.03847

[18] Hidasi, Balázs ; Karatzoglou, Alexandros ; Baltrunas, Linas ; Tikk, Domonkos: Session-based Recommendations with Recurrent Neural Networks. In: *arXiv:1511.06939 [cs]* (2016), März. – URL http://arxiv.org/abs/1511.06939. – Zugriffsdatum: 2020-07-03. – arXiv: 1511.06939

[19] Howard, Jeremy ; Ruder, Sebastian: Universal Language Model Fine-tuning for Text Classification. In: *arXiv:1801.06146 [cs, stat]* (2018), Mai. – URL http://arxiv.org/abs/1801.06146. – Zugriffsdatum: 2022-03-10. – arXiv: 1801.06146

[20] Ian Goodfellow ; Yoshua Bengio ; Aaron Courville: *Deep Learning*. MIT Press, 2016. – URL https://www.deeplearningbook.org/. – Zugriffsdatum: 2018-09-21

[21] Jannach, Dietmar ; Zanker, Markus ; Felfernig, Alexander ; Friedrich, Gerhard: *Recommender Systems*. 1. Cambridge University Press, 2010. – https://doi.org/10.1017/CBO9780511763113. – ISBN 978-0-521-49336-9

[22] Joshi, Mandar ; Chen, Danqi ; Liu, Yinhan ; Weld, Daniel S. ; Zettlemoyer, Luke ; Levy, Omer: SpanBERT: Improving Pre-training by Representing and Predicting Spans. In: *arXiv:1907.10529 [cs]* (2020), Januar. – URL http://arxiv.org/abs/1907.10529. – Zugriffsdatum: 2022-03-18. – arXiv: 1907.10529

[23] Jozefowicz, Rafal ; Vinyals, Oriol ; Schuster, Mike ; Shazeer, Noam ; Wu, Yonghui: Exploring the Limits of Language Modeling. In: *arXiv:1602.02410 [cs]* (2016), Februar. – URL http://arxiv.org/abs/1602.02410. – Zugriffsdatum: 2022-03-09. – arXiv: 1602.02410

[24] Kang, Wang-Cheng ; McAuley, Julian: Self-Attentive Sequential Recommendation. In: *arXiv:1808.09781 [cs]* (2018), August. – URL http://arxiv.org/abs/1808.09781. – Zugriffsdatum: 2022-03-04. – arXiv: 1808.09781

[25] Kudo, Taku ; Richardson, John: SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In: *arXiv:1808.06226 [cs]* (2018), August. – URL http://arxiv.org/abs/1808.06226. – Zugriffsdatum: 2022-03-26. – arXiv: 1808.06226

[26] Labeau, Matthieu: *Neural language models: Dealing with large vocabularies*, Université Paris Saclay (COmUE), Dissertation, September 2018. – URL https://tel.archives-ouvertes.fr/tel-02054671/. – https://tel.archives-ouvertes.fr/tel-02054671/file/75929_LABEAU_2018_archivage.pdf

[27] Lan, Zhenzhong ; Chen, Mingda ; Goodman, Sebastian ; Gimpel, Kevin ; Sharma, Piyush ; Soricut, Radu: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In: *arXiv:1909.11942 [cs]* (2020), Februar. – URL http://arxiv.org/abs/1909.11942. – Zugriffsdatum: 2022-03-18. – arXiv: 1909.11942

[28] Lange, Timo: Building a data pipeline for News Recommendation with Deep Learning. URL https://users.informatik.haw-hamburg.de/~ubicomp/

projekte/master2020-proj/lange.pdf, Juni 2020. – Forschungsbericht. – 27 S

[29] Lange, Timo: Reproduction of a Deep Learning Recommendation Model for usage as News Recommendation System. Hamburg, Oktober 2021. – Forschungsbericht. – 19 S. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2021-proj/lange.pdf

[30] Liu, Yinhan ; Ott, Myle ; Goyal, Naman ; Du, Jingfei ; Joshi, Mandar ; Chen, Danqi ; Levy, Omer ; Lewis, Mike ; Zettlemoyer, Luke ; Stoyanov, Veselin: RoBERTa: A Robustly Optimized BERT Pretraining Approach. In: *arXiv:1907.11692 [cs]* (2019), Juli. – URL http://arxiv.org/abs/1907.11692. – Zugriffsdatum: 2022-03-18. – arXiv: 1907.11692

[31] Manning, Christopher D. ; Raghavan, Prabhakar ; Schütze, Hinrich: Introduction to Information Retrieval. (2009), S. 569

[32] Okura, Shumpei ; Tagami, Yukihiro ; Ono, Shingo ; Tajima, Akira: Embedding-based News Recommendation for Millions of Users. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM, 2017 (KDD '17), S. 1933–1942. – URL http://doi.acm.org/10.1145/3097983.3098108. – Zugriffsdatum: 2018-05-20. – ISBN 978-1-4503-4887-4

[33] Peters, Matthew E. ; Neumann, Mark ; Iyyer, Mohit ; Gardner, Matt ; Clark, Christopher ; Lee, Kenton ; Zettlemoyer, Luke: Deep contextualized word representations. In: *arXiv:1802.05365 [cs]* (2018), März. – URL http://arxiv.org/abs/1802.05365. – Zugriffsdatum: 2022-03-15. – arXiv: 1802.05365

[34] Petroni, Fabio ; Rocktäschel, Tim ; Riedel, Sebastian ; Lewis, Patrick ; Bakhtin, Anton ; Wu, Yuxiang ; Miller, Alexander: Language Models as Knowledge Bases? In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China : Association for Computational Linguistics, 2019, S. 2463–2473. – URL https://www.aclweb.org/anthology/D19-1250. – Zugriffsdatum: 2022-04-13

[35] Radford, Alec ; Narasimhan, Karthik ; Salimans, Tim ; Sutskever, Ilya: Improving Language Understanding by Generative Pre-Training. (2018), S. 12

[36] Radford, Alec ; Wu, Jeffrey ; Child, Rewon ; Luan, David ; Amodei, Dario ; Sutskever, Ilya: Language Models are Unsupervised Multitask Learners. (2019), S. 24

[37] Rae, Jack W. ; Borgeaud, Sebastian ; Cai, Trevor ; Millican, Katie ; Hoffmann, Jordan ; Song, Francis ; Aslanides, John ; Henderson, Sarah ; Ring, Roman ; Young, Susannah ; Rutherford, Eliza ; Hennigan, Tom ; Menick, Jacob ; Cassirer, Albin ; Powell, Richard ; Driessche, George van d. ; Hendricks, Lisa A. ; Rauh, Maribeth ; Huang, Po-Sen ; Glaese, Amelia ; Welbl, Johannes ; Dathathri, Sumanth ; Huang, Saffron ; Uesato, Jonathan ; Mellor, John ; Higgins, Irina ; Creswell, Antonia ; McAleese, Nat ; Wu, Amy ; Elsen, Erich ; Jayakumar, Siddhant ; Buchatskaya, Elena ; Budden, David ; Sutherland, Esme ; Simonyan, Karen ; Paganini, Michela ; Sifre, Laurent ; Martens, Lena ; Li, Xiang L. ; Kuncoro, Adhiguna ; Nematzadeh, Aida ; Gribovskaya, Elena ; Donato, Domenic ; Lazaridou, Angeliki ; Mensch, Arthur ; Lespiau, Jean-Baptiste ; Tsimpoukelli, Maria ; Grigorev, Nikolai ; Fritz, Doug ; Sottiaux, Thibault ; Pajarskas, Mantas ; Pohlen, Toby ; Gong, Zhitao ; Toyama, Daniel ; d'Autume, Cyprien de M. ; Li, Yujia ; Terzi, Tayfun ; Mikulik, Vladimir ; Babuschkin, Igor ; Clark, Aidan ; Casas, Diego de L. ; Guy, Aurelia ; Jones, Chris ; Bradbury, James ; Johnson, Matthew ; Hechtman, Blake ; Weidinger, Laura ; Gabriel, Iason ; Isaac, William ; Lockhart, Ed ; Osindero, Simon ; Rimell, Laura ; Dyer, Chris ; Vinyals, Oriol ; Ayoub, Kareem ; Stanway, Jeff ; Bennett, Lorrayne ; Hassabis, Demis ; Kavukcuoglu, Koray ; Irving, Geoffrey: Scaling Language Models: Methods, Analysis & Insights from Training Gopher. In: *arXiv:2112.11446 [cs]* (2022), Januar. – URL http://arxiv.org/abs/2112.11446. – Zugriffsdatum: 2022-03-14. – arXiv: 2112.11446

[38] Raffel, Colin ; Shazeer, Noam ; Roberts, Adam ; Lee, Katherine ; Narang, Sharan ; Matena, Michael ; Zhou, Yanqi ; Li, Wei ; Liu, Peter J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. In: *arXiv:1910.10683 [cs, stat]* (2020), Juli. – URL http://arxiv.org/abs/1910.10683. – Zugriffsdatum: 2022-03-19. – arXiv: 1910.10683

[39] Reimers, Nils ; Gurevych, Iryna: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: *arXiv:1908.10084 [cs]* (2019), August. – URL

http://arxiv.org/abs/1908.10084. – Zugriffsdatum: 2022-03-15. – arXiv: 1908.10084

[40] Rendle, Steffen ; Freudenthaler, Christoph ; Gantner, Zeno ; Schmidt-Thieme, Lars: BPR: Bayesian Personalized Ranking from Implicit Feedback. (2009), S. 10

[41] Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.): *Recommender Systems Handbook*. Boston, MA : Springer US, 2015. – URL http://link.springer.com/10.1007/978-1-4899-7637-6. – Zugriffsdatum: 2022-02-08. – ISBN 978-1-4899-7636-9 978-1-4899-7637-6

[42] Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Boston, MA : Springer US, 2011. – URL http://link.springer.com/10.1007/978-0-387-85820-3. – Zugriffsdatum: 2022-02-08. – ISBN 978-0-387-85819-7 978-0-387-85820-3

[43] Salvado, André S.: Language Agnostic News Recommendations Using Deep Learning Based Causal Transformer Decoder. In: *Natural Language Processing* (2022), Februar, S. 72

[44] Sanh, Victor ; Debut, Lysandre ; Chaumond, Julien ; Wolf, Thomas: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In: *arXiv:1910.01108 [cs]* (2020), Februar. – URL http://arxiv.org/abs/1910.01108. – Zugriffsdatum: 2022-03-12. – arXiv: 1910.01108

[45] Schafer, J. B. ; Konstan, Joseph A. ; Riedl, John: E-Commerce Recommendation Applications. In: Kohavi, Ron (Hrsg.) ; Provost, Foster (Hrsg.): *Applications of Data Mining to Electronic Commerce*. Boston, MA : Springer US, 2001, S. 115–153. – URL http://link.springer.com/10.1007/978-1-4615-1627-9_6. – Zugriffsdatum: 2022-02-24. – ISBN 978-1-4613-5648-6 978-1-4615-1627-9

[46] So, David R. ; Mańke, Wojciech ; Liu, Hanxiao ; Dai, Zihang ; Shazeer, Noam ; Le, Quoc V.: Primer: Searching for Efficient Transformers for Language Modeling. In: *arXiv:2109.08668 [cs]* (2022), Januar. – URL http://arxiv.org/abs/2109.08668. – Zugriffsdatum: 2022-03-14. – arXiv: 2109.08668

[47] Souza Pereira Moreira, Gabriel de ; Rabhi, Sara ; Lee, Jeong M. ; Ak, Ronay ; Oldridge, Even: Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation. In: *Fifteenth ACM Conference on Recommender Systems*. Amsterdam Netherlands : ACM, September 2021, S. 143–153. – URL https://dl.acm.org/doi/10.1145/3460231.3474255. – Zugriffsdatum: 2022-03-07. – ISBN 978-1-4503-8458-2

[48] Sun, Fei ; Liu, Jun ; Wu, Jian ; Pei, Changhua ; Lin, Xiao ; Ou, Wenwu ; Jiang, Peng: BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In: *arXiv:1904.06690 [cs]* (2019), August. – URL http://arxiv.org/abs/1904.06690. – Zugriffsdatum: 2022-03-04. – arXiv: 1904.06690

[49] Sun, Yu ; Wang, Shuohuan ; Feng, Shikun ; Ding, Siyu ; Pang, Chao ; Shang, Junyuan ; Liu, Jiaxiang ; Chen, Xuyi ; Zhao, Yanbin ; Lu, Yuxiang ; Liu, Weixin ; Wu, Zhihua ; Gong, Weibao ; Liang, Jianzhong ; Shang, Zhizhou ; Sun, Peng ; Liu, Wei ; Ouyang, Xuan ; Yu, Dianhai ; Tian, Hao ; Wu, Hua ; Wang, Haifeng: ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation. In: *arXiv:2107.02137 [cs]* (2021), Juli. – URL http://arxiv.org/abs/2107.02137. – Zugriffsdatum: 2022-03-15. – arXiv: 2107.02137

[50] Sun, Yu ; Wang, Shuohuan ; Li, Yukun ; Feng, Shikun ; Tian, Hao ; Wu, Hua ; Wang, Haifeng: ERNIE 2.0: A Continual Pre-training Framework for Language Understanding. In: *arXiv:1907.12412 [cs]* (2019), November. – URL http://arxiv.org/abs/1907.12412. – Zugriffsdatum: 2022-03-15. – arXiv: 1907.12412

[51] Tang, Jiaxi ; Wang, Ke: Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In: *arXiv:1809.07426 [cs]* (2018), September. – URL http://arxiv.org/abs/1809.07426. – Zugriffsdatum: 2022-03-29. – arXiv: 1809.07426

[52] Torfi, Amirsina ; Shirvani, Rouzbeh A. ; Keneshloo, Yaser ; Tavaf, Nader ; Fox, Edward A.: Natural Language Processing Advancements By Deep Learning: A Survey. In: *arXiv:2003.01200 [cs]* (2021), Februar. – URL http://arxiv.org/abs/2003.01200. – Zugriffsdatum: 2022-03-10. – arXiv: 2003.01200

[53] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: Attention Is All

You Need. In: *arXiv:1706.03762 [cs]* (2017), Dezember. – URL http://arxiv.org/abs/1706.03762. – Zugriffsdatum: 2020-07-29. – arXiv: 1706.03762

[54] Wang, Hongwei ; Zhang, Fuzheng ; Xie, Xing ; Guo, Minyi: DKN: Deep Knowledge-Aware Network for News Recommendation. In: *Proceedings of the 2018 World Wide Web Conference*. Republic and Canton of Geneva, Switzerland : International World Wide Web Conferences Steering Committee, 2018 (WWW '18), S. 1835–1844. – URL https://doi.org/10.1145/3178876.3186175. – Zugriffsdatum: 2018-07-26. – ISBN 978-1-4503-5639-8

[55] Wang, Shoujin ; Cao, Longbing ; Wang, Yan ; Sheng, Quan Z. ; Orgun, Mehmet A. ; Lian, Defu: A Survey on Session-based Recommender Systems. In: *ACM Computing Surveys* 54 (2022), September, Nr. 7, S. 1–38. – URL https://dl.acm.org/doi/10.1145/3465401. – Zugriffsdatum: 2022-03-02. – ISSN 0360-0300, 1557-7341

[56] Wang, Wei ; Bi, Bin ; Yan, Ming ; Wu, Chen ; Bao, Zuyi ; Xia, Jiangnan ; Peng, Liwei ; Si, Luo: StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding. In: *arXiv:1908.04577 [cs]* (2019), September. – URL http://arxiv.org/abs/1908.04577. – Zugriffsdatum: 2022-03-15. – arXiv: 1908.04577

[57] Wu, Chuhan ; Wu, Fangzhao ; Ge, Suyu ; Qi, Tao ; Huang, Yongfeng ; Xie, Xing: Neural News Recommendation with Multi-Head Self-Attention. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China : Association for Computational Linguistics, 2019, S. 6388–6393. – URL https://www.aclweb.org/anthology/D19-1671. – Zugriffsdatum: 2022-02-28

[58] Wu, Chuhan ; Wu, Fangzhao ; Huang, Yongfeng ; Xie, Xing: Personalized News Recommendation: Methods and Challenges. In: *arXiv:2106.08934 [cs]* (2022), Februar. – URL http://arxiv.org/abs/2106.08934. – Zugriffsdatum: 2022-02-25. – arXiv: 2106.08934

[59] Wu, Chuhan ; Wu, Fangzhao ; Qi, Tao ; Huang, Yongfeng: Empowering News Recommendation with Pre-trained Language Models. In: *arXiv:2104.07413 [cs]* (2021), April. – URL http://arxiv.org/abs/2104.07413. – Zugriffsdatum: 2021-10-22. – arXiv: 2104.07413

[60] Wu, Chuhan ; Wu, Fangzhao ; Qi, Tao ; Huang, Yongfeng: Is News Recommendation a Sequential Recommendation Task? In: *arXiv:2108.08984 [cs]* (2021), August. – URL http://arxiv.org/abs/2108.08984. – Zugriffsdatum: 2022-04-14. – arXiv: 2108.08984

[61] Wu, Chuhan ; Wu, Fangzhao ; Qi, Tao ; Huang, Yongfeng ; Xie, Xing: Fastformer: Additive Attention Can Be All You Need. In: *arXiv:2108.09084 [cs]* (2021), September. – URL http://arxiv.org/abs/2108.09084. – Zugriffsdatum: 2021-10-19. – arXiv: 2108.09084

[62] Wu, Fangzhao ; Qiao, Ying ; Chen, Jiun-Hung ; Wu, Chuhan ; Qi, Tao ; Lian, Jianxun ; Liu, Danyang ; Xie, Xing ; Gao, Jianfeng ; Wu, Winnie ; Zhou, Ming: MIND: A Large-scale Dataset for News Recommendation. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online : Association for Computational Linguistics, 2020, S. 3597–3606. – URL https://www.aclweb.org/anthology/2020.acl-main.331. – Zugriffsdatum: 2021-10-19

[63] Wu, Liwei ; Li, Shuqing ; Hsieh, Cho-Jui ; Sharpnack, James: SSE-PT: Sequential Recommendation Via Personalized Transformer. In: *Fourteenth ACM Conference on Recommender Systems*. Virtual Event Brazil : ACM, September 2020, S. 328–337. – URL https://dl.acm.org/doi/10.1145/3383313.3412258. – Zugriffsdatum: 2022-03-05. – ISBN 978-1-4503-7583-2

[64] Yang, Zhilin ; Dai, Zihang ; Yang, Yiming ; Carbonell, Jaime ; Salakhutdinov, Ruslan ; Le, Quoc V.: XLNet: Generalized Autoregressive Pretraining for Language Understanding. In: *arXiv:1906.08237 [cs]* (2020), Januar. – URL http://arxiv.org/abs/1906.08237. – Zugriffsdatum: 2022-03-19. – arXiv: 1906.08237

[65] Yoav Goldberg: *SYNTHESIS LECTURES ON HUMAN LANGUAGE TECHNOLOGIES*. Bd. 37: *Neural Network Methods for Natural Language Processing*. Morgan & Claypool, 2017. – DOI 10.2200/S00762ED1V01Y201703HLT037. – ISBN 978-1-62705-298-6

[66] Zhang, Qi ; Li, Jingjie ; Jia, Qinglin ; Wang, Chuyuan ; Zhu, Jieming ; Wang, Zhaowei ; He, Xiuqiang: UNBERT: User-News Matching BERT for News Recommendation. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. Montreal, Canada : International Joint Confer-

ences on Artificial Intelligence Organization, August 2021, S. 3356–3362. – URL https://www.ijcai.org/proceedings/2021/462. – Zugriffsdatum: 2022-03-04. – ISBN 978-0-9992411-9-6

[67] Zhang, Rongzhi ; Gu, Yulong ; Shen, Xiaoyu ; Su, Hui: Knowledge-enhanced Session-based Recommendation with Temporal Transformer. In: *arXiv:2112.08745 [cs]* (2021), Dezember. – URL http://arxiv.org/abs/2112.08745. – Zugriffsdatum: 2022-03-05. – arXiv: 2112.08745

[68] Zhang, Shuai ; Yao, Lina ; Sun, Aixin ; Tay, Yi: Deep Learning based Recommender System: A Survey and New Perspectives. In: *ACM Computing Surveys* 1 (2019), Juli, Nr. 1, S. 35

[69] Zhang, Yongfeng ; Chen, Xu: Explainable Recommendation: A Survey and New Perspectives. In: *Foundations and Trends® in Information Retrieval* 14 (2020), Nr. 1, S. 1–101. – URL http://www.nowpublishers.com/article/Details/INR-066. – Zugriffsdatum: 2022-03-05. – ISSN 1554-0669, 1554-0677

[70] Zhang, Yuanxing ; Zhao, Pengyu ; Guan, Yushuo ; Chen, Lin ; Bian, Kaigui ; Song, Lingyang ; Cui, Bin ; Li, Xiaoming: Preference-Aware Mask for Session-Based Recommendation with Bidirectional Transformer. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mai 2020, S. 3412–3416. – ISSN: 2379-190X

[71] Zhang, Zhengyan ; Han, Xu ; Liu, Zhiyuan ; Jiang, Xin ; Sun, Maosong ; Liu, Qun: ERNIE: Enhanced Language Representation with Informative Entities. In: *arXiv:1905.07129 [cs]* (2019), Juni. – URL http://arxiv.org/abs/1905.07129. – Zugriffsdatum: 2022-03-15. – arXiv: 1905.07129

[72] Zheng, Guanjie ; Zhang, Fuzheng ; Zheng, Zihan ; Xiang, Yang ; Yuan, Nicholas J. ; Xie, Xing ; Li, Zhenhui: DRN: A Deep Reinforcement Learning Framework for News Recommendation. In: *Proceedings of the 2018 World Wide Web Conference*. Republic and Canton of Geneva, Switzerland : International World Wide Web Conferences Steering Committee, 2018 (WWW '18), S. 167–176. – URL https://doi.org/10.1145/3178876.3185994. – Zugriffsdatum: 2018-05-20. – ISBN 978-1-4503-5639-8

[73] Zheng, Lei ; Lu, Chun-Ta ; He, Lifang ; Xie, Sihong ; He, Huang ; Li, Chaozhuo ; Noroozi, Vahid ; Dong, Bowen ; Yu, Philip S.: MARS: Memory Attention-Aware

Recommender System. In: *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Oktober 2019, S. 11–20

A Appendix

A.1 Training Model Implementation

"The implementation of the model is done via *Keras* and *Tensorflow*, where the high-level *Keras functional API* is used when possible and lower level *Tensorflow* operations when necessary. Figure 3.7 shows the operational graph generated by *Tensorboard*. The big outer box includes the whole MARS model, where the output to the loss function is shown at the top and the input at the bottom, represented by an arrow. The right light purple box named *Model_ context_ items* is the CNN which computes the representation of the items liked and not liked from the input tuple. Both items are processed by two exact same CNNs sharing their weights and are thus depicted by just one sub model. On the left side in a light yellow box, called *time_ distributed*, the item representations for the items in the user history are computed, whose result is the memory component shown in figure 3.5. The *time_ distributed* layer is a built-in *Keras* layer which applies an passed layer to the data stepwise. In this case, the passed layer is the CNN to generate the users history items representation called *Model_ user_ items*, which is applied to each item in a users history. To eliminate any confusion, in *Keras*, models can also be treated as layers. Finally, with the memory component and the liked/not liked item representations, the attention for the two items is computed from which the user representations for the liked and not liked items are derived. Both, item and user representations are concatenated and feed to the loss function."[29]

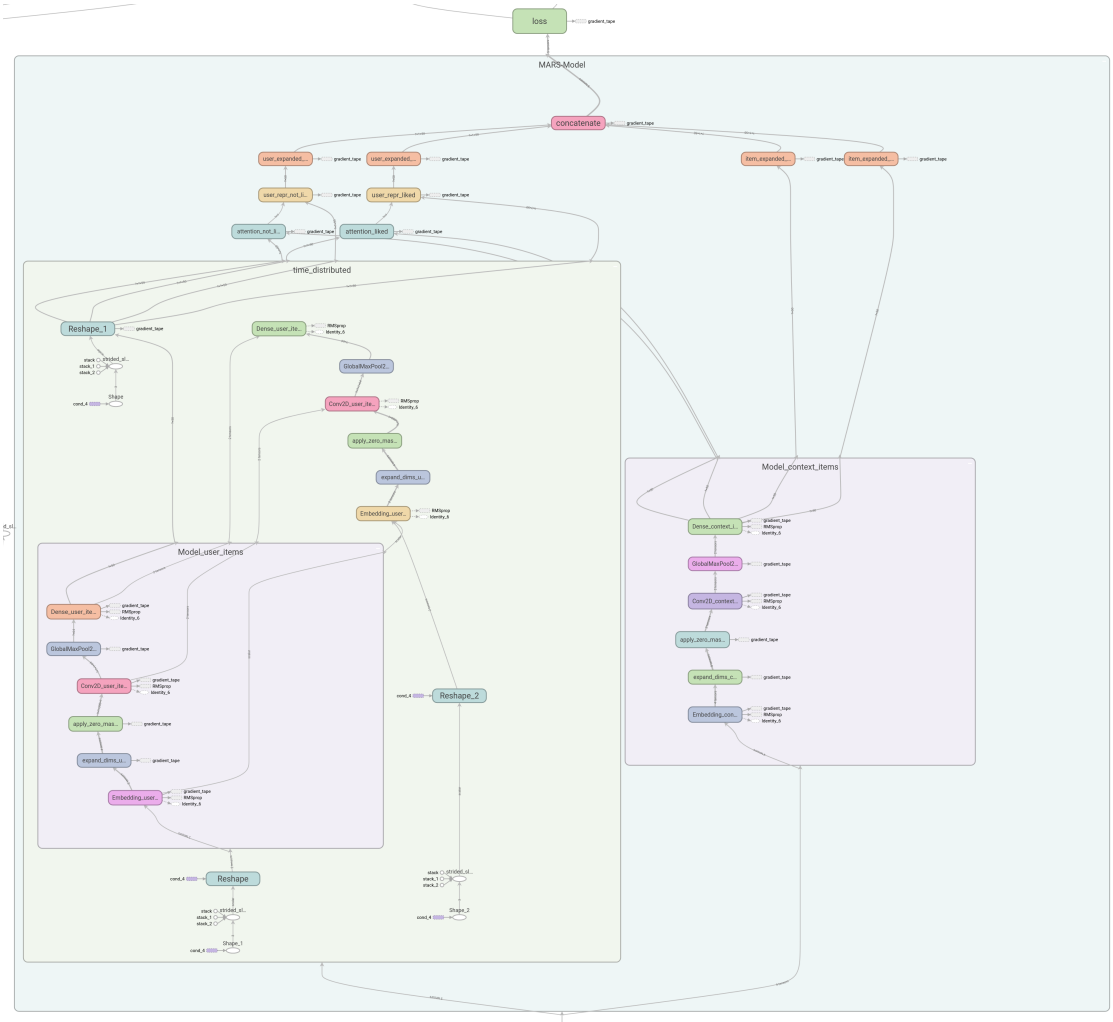


Figure A.1: *Tensorboard* graph of MARS implementation [29]

A.2 Dataset Tables

Further analysis of the dataset and detailed explanation of the data fields can be found in [43].

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
user_id	string	56199311	100.00				
session_id	string	167711970	100.00				
session_referer_medium	string	7	100.00				
session_referer_source	string	87	36.64				

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
geo							
.city	string	55970	92.73				
.country	string	228	99.80				
.latitude	double	71819	99.91				
.longitude	double	78556	99.91				
.region	string	1408	93.51				
.region_name	string	3274	93.51				
.timezone	string	348	99.91				
.zipcode	string	55411	93.35				
browser							
.doc_width	int64	6408	100.00	0.00	862.87	84731896.00	12213.21
.doc_height	int64	133135	100.00	0.00	12154.72	82113264.00	28841.89
.view_width	int64	4310	100.00	0.00	813.75	45008.00	581.51
.view_height	int64	17825	100.00	0.00	767.59	571698.00	370.66
os							
.family	string	0	0.00				
.manufacturer	string	0	0.00				
.name	string	0	0.00				
.timezone	string	83	100.00				
device							
.browser_engine	string	0	0.00				
.type	string	0	0.00				
.is_mobile	bool	0	0.00				
useragent							
.device_class	string	13	97.77				
.device_name	string	10616	97.77				
.device_brand	string	938	97.77				
.agent_class	string	7	97.77				
.agent_name	string	558	97.77				
.agent_version	string	17226	97.77				
.operating_system_class	string	8	97.77				
.operating_system_name	string	52	97.77				
.operating_system_version	string	648	97.77				
.webview_app_name	string	185	13.01				
.layout_engine_class	string	4	97.77				
publisher_id	string	10	100.00				
portal_id	string	17	100.00				
page_view_id	string	312908725	100.00				
url_ebene_1	string	3272	76.82				
url_ebene_2	string	69908	49.73				
url_ebene_3	string	89921	17.01				

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
referer_url_ebene_1	string	3912	20.25				
referer_url_ebene_2	string	40686	11.99				
referer_url_ebene_3	string	26383	3.53				
user_type	string	3	100.00				
x_scroll_pct	double	101	99.95	0.00	98.66	100.00	8.73
y_scroll_pct	double	101	99.99	0.00	40.74	100.00	28.31
x_scroll_pct_min	double	1	99.95	0.00	0.00	0.00	0.00
y_scroll_pct_min	double	35	99.99	0.00	0.00	60.00	0.01
time_engaged_in_s	int64	1036	100.00	0.00	30.88	489825.00	53.06
article_type	string	3	60.53				
is_paywall	bool	2	92.39	0.00	0.26	1.00	0.44
content_type	string	5	100.00				
user_engagement_segment	string	7	61.90				
article_drive_id	string	823944	48.14				
page_view_start_local	timestamp[us]	307171178	100.00				

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
page_view_end_local	timestamp[us]	307862602	100.00				
pv_article_completion							
.completion_time_in_s	double	2710	48.14				
.fraction_article_read	double	79904	48.14				
.fraction_article_read_binned	double	21	48.14				
.completion	bool	2	48.14				
completion_time_in_s	double	2710	48.14	0.40	199.73	11932.40	254.00
fraction_article_read	double	79904	48.14	0.00	0.24	1.00	0.28
completion	bool	2	48.14	0.00	0.06	1.00	0.24

Table A.1: Schickler News Recommendation dataset: Statistics about pageviews.

Field name	Type	Unique	% Not Null	Words/Values			
				min	avg	max	Std.
publisher_id	string	10	100.00				
article_drive_id	string	823944	100.00				
article_header	string	755922	100.00	0.00	6.35	29.00	2.60
article_teaser	string	738363	100.00	0.00	24.08	1485.00	15.38
article_full_text	string	821750	100.00	0.00	398.35	30642.00	279.24
is_plus_article	bool	2	97.77	0.00	0.19	1.00	0.39
article_dpa_id	string	47290	11.92				
is_dpa	bool	2	100.00	0.00	0.12	1.00	0.32
published_at_local	timestamp[us]	542925	100.00				
modified_at_local	timestamp[us]	736714	100.00				
pad_pleasure	double	302096	37.18				
pad_arousal	double	302353	37.18				
pad_dominance	double	302099	37.18				
preview_pad_pleasure	double	359778	46.77				
preview_pad_arousal	double	360095	46.77				
preview_pad_dominance	double	358928	46.77				
emo_aerger	double	293978	37.18				
emo_erwarten	double	299558	37.18				
emo_ekel	double	280939	37.18				
emo_furcht	double	298868	37.18				
emo_freude	double	300252	37.18				
emo_traurigkeit	double	295867	37.18				
emo_ueberraschung	double	293821	37.18				
emo_vertrauen	double	301083	37.18				
article_preview_emotion							
.emo_aerger	double	127817	46.77				
.emo_erwarten	double	172357	46.77				
.emo_ekel	double	79353	46.77				
.emo_furcht	double	165101	46.77				
.emo_freude	double	231700	46.77				
.emo_traurigkeit	double	132540	46.77				
.emo_ueberraschung	double	114664	46.77				
.emo_vertrauen	double	226332	46.77				
preview_emo_aerger	double	127817	46.77				
preview_emo_erwarten	double	172357	46.77				
preview_emo_ekel	double	79353	46.77				
preview_emo_furcht	double	165101	46.77				
preview_emo_freude	double	231700	46.77				
preview_emo_traurigkeit	double	132540	46.77				
preview_emo_ueberraschung	double	114664	46.77				
preview_emo_vertrauen	double	226332	46.77				
article_header_contains_quote	bool	2	100.00				
article_header_contains_question	bool	2	100.00				
article_header_contains_doppelpunkt	bool	2	100.00				
article_header_contains_pronoun_writer	bool	2	100.00				
article_header_contains_pronoun_reader	bool	2	100.00				
article_preview_contains_quote	bool	2	100.00				
article_preview_contains_question	bool	2	100.00				
article_preview_contains_doppelpunkt	bool	2	100.00				
article_preview_contains_pronoun_writer	bool	2	100.00				
article_preview_contains_pronoun_reader	bool	2	100.00				
topic	string	17	71.39				
locality	string	4	71.60				
newstype	string	3	70.12				
genre	string	8	59.66				

Table A.2: Schickler News Recommendation dataset: Statistics about articles.

A.3 News Article Example

Header: Droht Regensburg eine Rattenplage?

Teaser: Restaurants dürfen derzeit nur Essen zum Mitnehmen anbieten. In manchen bayerischen Orten führte das zu höherem Rattenbefall.

Full Text: Die Restaurants sind zu, die Menschen kaufen in der Corona-Zeit deshalb mehr Essen zum Mitnehmen. Dadurch entsteht mehr Müll, der Ratten anlockt. Städte und Gemeinden berichten vereinzelt davon, dass die Zahl der Tiere an Müllcontainern und der Kanalisation zugenommen habe. Wie sieht die Lage in Regensburg aus?Stadtsprecherin Katrin Butz gibt hier Entwarnung: „In Regensburg kann kein höherer Rattenbefall als in den früheren Jahren festgestellt werden." Andernorts in Bayern sieht das hingegen anders aus. So zum Beispiel in Volkach im Landkreis Kitzingen, wo die Verwaltung von einem „hausgemachten Problem" mit Ratten spricht. Die wichtigsten Maßnahmen zur VorbeugungWeil in der Kanalisation mehr Ratten festgestellt wurden und die Tiere Infektionskrankheiten übertragen, gehen in der unterfränkischen Stadt Mitarbeiter der Kläranlage mit Giftköder gegen die Tiere vor. Die Verwaltung hatte die Bürger gebeten, keine Speisereste in der Toilette zu entsorgen oder in Parks und auf Spielplätzen wegzuwerfen.Speisereste und To-Go-Verpackungen in die dafür vorgesehenen Tonnen und Abfallbehälter zu entsorgen sei die wichtigste Maßnahme zur Vorbeugung, heißt es seitens der Stadt Regensburg. Auch sollten Wertstoffsäcke immer erst am Tag der Abholung vor dem Haus abgestellt und keine Essensreste auf dem Kompost entsorgt werden.Die bayerischen Kommunen sind gewappnetDie bayerischen Kommunen sind gewappnet, stellen aber keine flächendeckende Zunahme der Ratten-Population fest. Aus Nürnberg heißt es etwa, dass zwar mehr To-Go-Essen verkauft werde, aber grundsätzlich weniger Menschen in der Stadt unterwegs seien. „In der Regel wird das Essen daheim eingenommen", teilt ein Sprecher mit. Insofern lägen nicht mehr Essensreste herum. Zudem gehe die Stadt mit regelmäßigen Kontrollen gegen die Nager vor.München, Augsburg, Rosenheim, Landshut, Bamberg und Hof meldeten bei der Zunahme von Ratten ebenfalls: „Fehlanzeige". Die Stadt München teilt etwa mit, dass gelegentlich Spielplätze wegen Maßnahmen zur Rattenbekämpfung innerhalb des Stadtgebiets gesperrt werden müssten. Das sei aber seit jeher zum Schutz von Kleinkindern notwendig. Einen Zusammenhang mit der Pandemie gebe es nicht. (mit Material der dpa)

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original