LEUPHANA
UNIVERSITÄT LÜNEBURG

Major Management & Data Science
Institute of Information Systems
Leuphana University Lüneburg

Masterthesis

Unbiased Neural Ranking Models for Product Search

Submitted by Laurin Luttmann
Student ID no.: 3038590
Email: laurin.luttmann@gmail.com
Address: Gorch-Fock-Str. 1, 20357 Hamburg

1. Reviewer Prof. Dr. Ulf Brefeld
Institute of Information Systems
Leuphana University Lüneburg

2. Reviewer Prof. Dr. Burkhardt Funk
Institute of Information Systems
Leuphana University Lüneburg

06.09.2022

Abstract

Online shopping platforms and marketplaces offer vast amounts of products. Amazon, the world's largest e-commerce company, has a product portfolio of 12 million products and a total of 350 million different products are offered on its marketplace. The growth of such platforms drastically increased the need for proper search engines. Such systems comprise a retrieval part, which selects all items of the shop that fit a user's search intent, issued by a text-based query, and a ranking part that sorts all retrieved products in descending order of their relevance. Given the immense number of available products in modern online shops, the retrieval set is usually still too large for a user to effectively navigate through it, making the ranking an inevitably important part of the customer experience. This work will investigate the means of learning an optimal ranking using past user interactions with the search results. However, unlike in web search, where the primary goal is to increase the number of clicks on search results, the notion of an optimal ranking is less evident in e-commerce. Different users of online shops have different intents. While some users want to order a specific product immediately, others want to get inspired without having an intent to purchase. Therefore, it is crucial to incorporate the manifold customer signals prevalent in the search logs of online shops, like clicks, orders, dwell-time and "add-to-basket" to serve diverse customer needs and optimize different KPIs. Nevertheless, using such signals directly as training targets will make the model biased towards the current ranking due to the position bias. Since people tend to click on products ranked higher on the result page of the search engine, naively using such data would make the model simply learn the current ranking instead of the product relevance. Therefore, this work will develop a framework to learn an unbiased ranking model from multiple relevance signals. Neural networks define the hypothesis space of the model due to their ability to process raw textual information like user queries and document descriptions, which pose the most important pieces of information for the ranking task. To test the proposed method, a large-scale online A/B test is conducted.

Keywords — Learning-to-Rank, counterfactual LTR, e-commerce, multi-task learning, neural networks, online A/B testing

Contents

List of Figures

List of Tables

1 Introduction

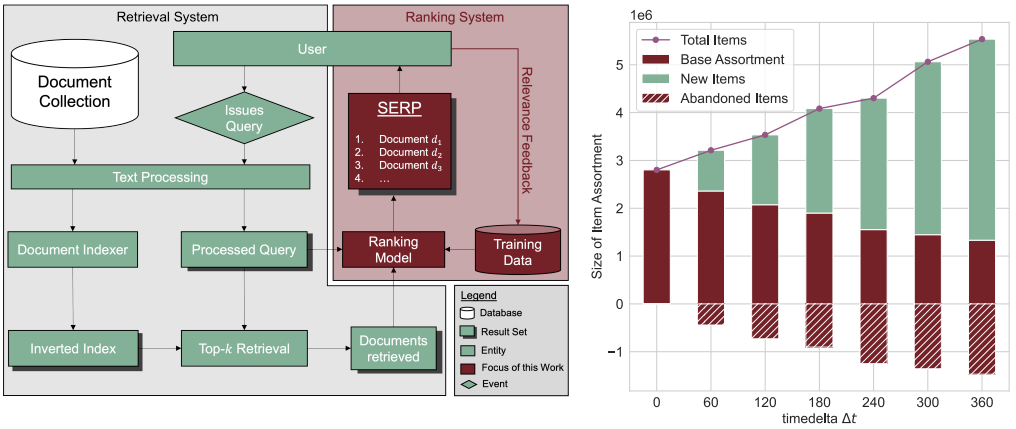
> *With the fast growth of the Internet, [...] the search engine has become an essential tool for many people*
>
> — **Tie-Yan Liu**
> (Researcher at Microsoft)

The internet is an ever-expanding collection of documents like web pages, videos or images. While the number of websites was around 15 million in the year 2000, the internet has grown to more than 100 million websites just seven years later [Liu09]. Today, the world's largest search engine, Google, claims to have information from around 100 billion websites in their index, which contains only the searchable web [Goo22a]. The growth of the internet itself has also had an impact on companies offering their services on the Word Wide Web. Online shops and e-commerce companies are fast-growing phenomena, not just since the Corona pandemic. Especially through recent tendencies to platformization, which result in the agglomeration of many vendors and their products into a single online shop, almost any product can be found and purchased online, making the internet a convenient alternative to shopping malls and city centers.

Just like the internet itself, online shops rely on search engines which help users to effectively navigate through these vast collections of products. Figure 1.1a shows a typical high-level search engine architecture, as described, for example, by Baeza-Yates and Ribeiro-Neto (1999). As seen from the figure, a search engine's components can be subdivided into a retrieval and a ranking part. The retrieval system has access to the entire document collection and stores its information, such as document header and body, after some text processing in an index structure, for example an inverted index. To search the document collection, the user specifies his/her intent through a (text-based) query and sends it to the search engine. The query runs through the same text processing and is compared against the index to retrieve a subset of the document collection.

The retrieved documents and the user query are then forwarded to the ranking system, which computes a relevance score for each retrieved document. The documents are then sorted in decreasing order of relevance to the query and presented to the user on the search engine result page (SERP).

The retrieval system is an important prerequisite for an adequate ranking since not retrieved documents will not be ranked and thus cannot appear on the SERP. On the other hand, when the retrieval system retrieves too many documents, the ranking process takes too long and consequently degrades the experience for the user, who typically expects a result within a few hundred milliseconds [Cam10]. Yet, this paper



(a) Typical Search Engine Architecture

(b) Evolution of □'s product assortment

Fig. 1.1.: Typical Search Engine Architecture (based on [BR99], left) and the Evolution of the Product Assortment over a horizon of 360 days (right)

will assume the retrieval system to be fixed and focus on the ranking of a retrieved collection of documents for a given query.

While very early search engines relied on heuristics for ranking products, machine learning has been applied to this task for more than 20 years [CC11]. The process of training a model for the ranking task is called Learning-to-Rank (LTR) [Ai19]. These models are usually trained using relevance feedback, which can be explicit or implicit. Explicit feedback is usually provided in the form of manually graded relevance labels for the various query-document pairs in the training data. Such human ratings are typically performed by experts since most users of search engines are reluctant to perform such tasks [DP08]. This makes human ratings extremely expensive and cumbersome to gather. Also, such expert ratings – if not performed continuously – only capture past relevance, which, especially in the e-commerce scenario with short product lifecycles, poses a significant drawback [LSR14]. Figure 1.1b illustrates these dynamics of the product assortment on the example of the XXX online shop over a period of one year. One can see that after 360 days, only half of the originally available products are still in the assortment, while new products account for more than three-quarters of the new product catalog.

Implicit feedback is represented by customer interactions with the search results, like clicks, dwell time or purchases, which can be traced through the logs of the website. Unlike human ratings, click-logs from users are cheap to collect, always up to date and also reflect the preferences of the users [Aga19b]. Moreover, for large e-commerce search engines, click-logs are available in huge quantities, which makes them especially useful when training very complex models for the ranking task like deep neural networks.

However, a major drawback of click data is that it is inherently biased towards the current ranking due to the position bias. Since higher-ranked items are more likely to be observed by the user than lower-ranked ones, they naturally receive more clicks regardless of their actual relevance. A model naively trained on such signals will consequently not learn the relevance of documents given a query but the order

in which they were displayed in the past [Ai18b]. This leads to endless feedback loops in which the same products are ranked at the top over and over again [SGR17]. Therefore, the position bias must be taken into account when learning a ranking function from click-log data.

This work will investigate the possibilities of leveraging implicit feedback gathered from click-logs for training a deep neural network for the product ranking at one of the largest e-commerce companies in Europe in terms of revenue (release version of this thesis will mask the identity of the company). It will therefore contribute to the current state of research in the field of counterfactual Learning-to-Rank and neural ranking with application to the e-commerce sector. A short introduction followed by related work in each of these areas is given in chapter 2. Drawing on these results, potential contributions of this master thesis are deduced. In chapter 3, the architectures and learning strategies for the ranking models that have been developed throughout this project are described. To evaluate the quality of these models, an extensive offline evaluation is conducted before using the models in an online A/B test. The results for both tests are presented in chapter 4. The thesis concludes with a critical appraisal and a conclusion in chapter 5.

2 Related Work

2.1 Learning-to-Rank

Learning-to-Rank is commonly defined as the task of fitting a ranking model using training data, such that the model can sort new objects according to their relevance given a context (query) [Liu09]. This ranking model is defined as a function f that maps documents $d \in D_q$ given a query q to a score s. The retrieved documents D_q are then sorted based on the scores given by the ranker to generate the ranking π [Hu19].

According to Liu (2009), the general Learning-to-Rank framework follows that of supervised learning. Hence, a training set is required consisting of a set of n training queries $Q = \{q_i\}_{i=1}^{n}$ and their corresponding documents $D_{q_i} = \{d_{ij}\}_{j=1}^{m_i}$, where m_i is the number of documents that have been retrieved for query q_i. Each of the resulting query-document pairs (q_i, d_{ij}) is represented by a k-dimensional feature vector $\mathbf{x}_{ij}$ and has a label y_{ij} associated with it, indicating the relevance of the document for that specific query. Features in Learning-to-Rank models can be categorized as one of three types [CC11]. First, query-dependent features describe the query regardless of the documents, for example the query length or how trending the query is. Likewise, document-specific features provide statistics about the document regardless of the query it appeared in, for example the popularity of the document measured by past clicks or its page rank [Pag99]. The third and, according to Chapelle and Chang (2011), most important type of features are the query and document dependent features, which describe the textual similarity between the query and the document. The most popular method of describing the similarity between a query and a document is the BM25, which builds on the idea of the TF-IDF statistic. The term frequency (TF) of a term t describes the number of its occurrences in a document d, normalized by the length of the document. The inverse document frequency (IDF) of a term t measures the specificity of a term with respect to the document collection in order to put more emphasis on rare terms. It is defined as the logarithm of the total number of documents in the collection N divided by the number of documents containing t. Formally, the TF-IDF of a term t and a document d is defined as:

$$\text{tfidf(t,d)} = \frac{tf_{td}}{\sum_{t' \in d} tf_{t'd}} \cdot \log \frac{N}{|d \in D : t \in d|} \tag{2.1}$$

where tf_{td} is number of times t occurs in d [RU11]. The TF-IDF score can be used as a query-document dependent feature in a ranking model by summing $\text{tfidf}(t_i, d)$ over all terms $t_1, ..., t_m$ of the query. That is the core idea of BM25, which in addition to the sum of TF-IDF scores uses some free parameters and additional statistics of the documents [Liu09].

Given the feature vectors and the corresponding relevance labels, the Learning-to-Rank model aims to find a mapping $f : \mathbb{R}^k \to \mathbb{R}$ from the feature space to the output space (the relevance labels) by minimizing the empirical risk:

$$\hat{\mathcal{R}}(f) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \mathcal{L}(f(\mathbf{x}_{ij}), y_{ij}) \tag{2.2}$$

Ideally, the minimization of the chosen loss function $\mathcal{L}$ also minimizes the ranking quality measures used to evaluate the ranking model, like Mean Reciprocal Rank (MRR), Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG), which will be described in detail in chapter 3.3.1. The peculiarity of these metrics is that they place stronger weight on the top of the ranked list while the contribution from a document fades as its rank increases [Bru19b]. Thereby, these metrics reflect the behavior of the users, who typically scan documents from top to bottom and are consequently less likely to examine documents at larger rank positions. To achieve this, ranking metrics consider the permutation π of documents after sorting according to the scores determined by $f(\cdot)$ rather than the scores directly. However, due to the sorting mechanism, they are not differentiable and consequently cannot be used in learning algorithms like stochastic gradient descent. Therefore, a lot of research in the field of LTR has been devoted to the invention of adequate loss functions.

Loss functions in LTR can be categorized as pointwise, pairwise or listwise [Hu19]. Pointwise learning algorithms directly approximate the relevance score of query-document pairs and thus can be represented by any regression or classification (in case of binary relevance labels) algorithm. A typical pointwise loss is the mean squared error for a real-valued output space or the cross-entropy loss for binary relevance labels. The pointwise LTR approach is considered inferior since these loss functions pay equal attention to every query-document pair in the training set and, consequently, are not well suited to approximate the above-mentioned ranking metrics. Moreover, pointwise methods ignore the competitive nature of ranked lists. The relevance of an item for a query does not only depend on the utility of the item itself but also on the utility of the other documents in the SERP. By approximating the score for one document at a time, these dynamics are not taken into account.

In order to capture the competition within a SERP, pairwise models take pairs of documents as input and try to predict the more relevant document in each case. Therefore, the relevance labels are transformed to reflect the pairwise preferences. Given the relevance labels y for each query-document pair, the pairwise preference for documents i and j can be defined as $y_{ij}^q = 2 \cdot \mathbb{1}(y_i \succ y_j) - 1$ [Cao07]. A pairwise loss function like RankNet [Bur05] then compares the ranking score difference $f(x_{qi}) - f(x_{qj})$ against y_{ij}^q. However, since this approach does only consider the relative order between two documents, the position of the documents in the final ranked list cannot be inferred. Consequently, pairwise methods cannot discount the influence of documents at greater rank positions either.

query	occurences
culotte	237
damenhose sommer	134
strandhose damen	128
7/8 hose damen sommer	92
sommerhose damen	83
culotte damen	72
damen hosen sommer	64
weite sommerhosen für damen	53

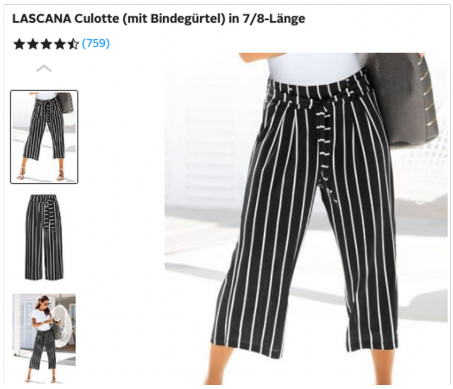


Fig. 2.1.: Different user queries and their frequencies (left) that all led to clicks on the same product (right)

Therefore, listwise LTR models take the entire set of documents associated with a query as input and try to optimize for IR metrics like NDCG directly. LambdaRank [BRL06] or LambdaMART [Bur10] for example still compute the gradient via a pairwise loss function, but scale it by the absolute change in NDCG when swapping the two documents in the ranked list. LambdaMART is currently considered state-of-the-art in LTR [ZK20; Bru19b].

2.2 Neural Ranking

Deep neural networks have led to significant improvements in many machine learning applications and constitute the state-of-the-art in domains like computer vision or natural language processing. A major advantage of such networks is their ability to learn the feature extractor ϕ from raw input (texts, images etc.). This typically leads to better representations (embeddings) of the objects of interest than with hand-crafted feature functions [Guo20a].

In the e-commerce scenario, vendors tend to describe products in professional language, while user queries are typically written in easy and everyday language [Yao21]. For example, users tend to formulate their search intent by using descriptive attributes for products, like “pants for the beach”, while vendors use technical terms like “culotte”. Figure 2.1 clarifies this phenomenon by showing the most used search queries that lead the customer to the example product. The technical term culotte is used in the product title, but only in a fraction of the queries. This vocabulary gap cannot be captured by traditional text matching functions like TF-IDF, which only perform exact text matching but do not attribute for synonyms, hypernyms and spelling errors [Bre18]. Deep neural networks on the other hand learn such semantic similarities from the raw input data presented to them when trained on massive amounts of data and thus should be more suitable for learning the ranking function for product search.

The first successful approach to using neural networks to learn query and document representations from raw text is the Deep Structured Semantic Model (DSSM)

proposed by Huang et al. (2013). The authors use a siamese network architecture to map the raw query text and the raw document description in a mutual space. Therefore, the siamese network uses the same weights for encoding query and document and in the end compares the similarity of the embeddings using the cosine similarity:

$$cos(q, d) = \frac{\phi(q) \cdot \phi(d)}{||\phi(q)|| \cdot ||\phi(d)||} . \tag{2.3}$$

The DSSM uses a mean pooling strategy over the token embeddings, which could be pre-trained by for example word2vec or learned from scratch. Then, fully connected units are used to produce the final query or document embedding. More recent approaches replace the pooling and fully connected part of the network with Convolutional Neural Networks [She14; Hu14] or Recurrent Neural Networks [Pal16] in order to preserve the word order.

The specificity of the DSSM and its variants is that it encodes query and documents individually, without modeling any kind of interaction between the two. This is a major drawback of these models since tokens that appear irrelevant in a document for one query could be important for another [MC18]. If for example one user searches for a dress in a specific color and another user for dresses in general, the color information in the document description is irrelevant for the latter but very important to the first user. Such dependencies cannot be captured by these so-called representation-focused models. Therefore, a different line in this research area uses interaction-based models, which first match different parts of the query with different parts of the document at a low level and then aggregate the partial matching scores to a final prediction [Nig19].

Figure 2.2 illustrates the general network architecture of representation-based matching models like DSSM and interaction-based models. Most interaction-based models use a matching matrix, consisting of the pairwise cosine similarities between each word embedding of the query and the document. MatchPyramid for example then uses this matching matrix as input for a Convolutional Network to predict the matching score [Pan16]. The Deep Relevance Matching Model (DRMM) discretizes the cosine similarities, which lie in the interval $[-1, 1]$, into H bins and calculates the number of pairwise cosine similarities falling in each interval. In order to stress the importance of exact text matches, a cosine similarity of exactly 1 falls into a separate bin. Each of the resulting "matching histograms" is utilized as input for an MLP to generate a matching score on query-token level. The individual query-token scores are then aggregated to produce the final relevance score [Guo16].

More recent studies use the transformer architecture developed by Vaswani et al. (2017), which enabled major breakthroughs on several NLP tasks. Cer et al. (2018) propose the Universal-Sentence-Encoder (USE), which uses the encoder block of the transformer architecture to compute context-aware representations of words that take into account all the other words in the sequence. These representations are then converted to a fixed-length sentence embedding by computing the element-wise sum of the token embeddings. And Nogueira and Cho (2020) use a pre-trained BERT

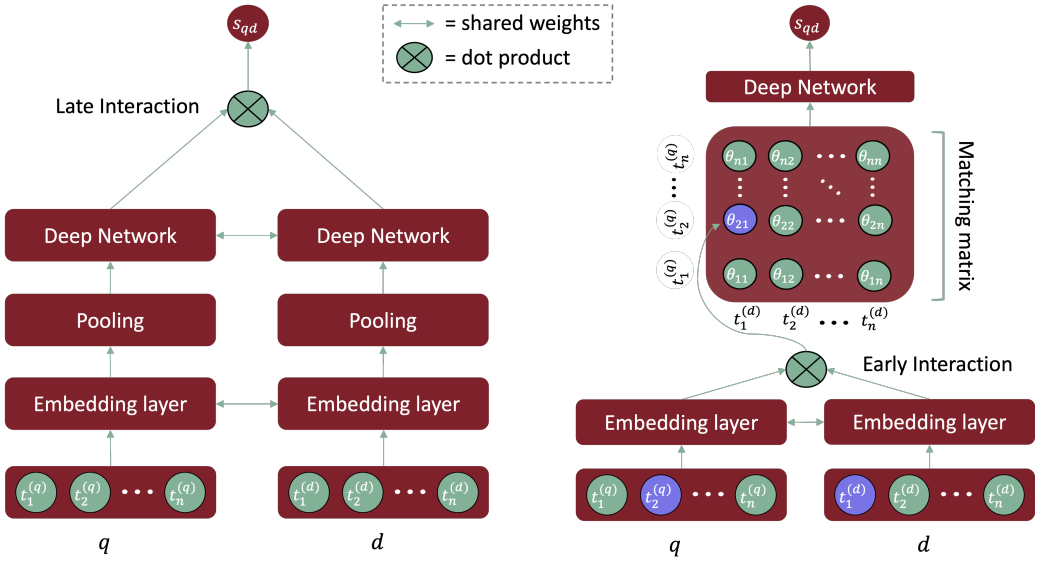


Fig. 2.2.: Representation- vs interaction-based matching models

model [Dev19] to directly obtain a similarity score between query and document. They feed query and document tokens jointly in the model by separating them using BERT's special [SEP] token. Then a classifier is built on top of the [CLS] token embedding to predict the matching score. While achieving state-of-the-art results on text matching benchmarks like MS MARCO [Baj18], these models are extremely resource expensive in practice [Yao21].

2.3 Learning from implicit Feedback

Deep learning models are known to converge only when exposed to a very large number of training examples due to their complexity. Under the traditional learning paradigm, where relevance judgments have to be gathered by human assessment for each query and its associated documents, learning a deep neural network is practically infeasible [Aga19b]. Instead, user interactions with the search results, which are recorded in the click-logs of the website, can be used to train such a ranking model. These interactions provide an implicit feedback from the users regarding the relevance of documents for the query they issued. By analyzing which documents the users click on in the presented ranking for a query, detailed information about the relevance of documents and consequently the optimal ranking can be extracted [DP08]. However, a fundamental problem is the position bias, which describes the phenomenon that a user is less likely to click on a document listed low in the ranking, independent of how relevant it is [Joa02]. Therefore, to get reliable relevance signals from clickthrough data, it is necessary to account for and model the influence of the rank position of a document on the user interactions with it.

Strategies for learning ranking functions from implicit feedback while accounting for the position bias are commonly referred to as counterfactual LTR. The first such approach was introduced by Joachims (2002), who developed a framework for learning

a ranking function from click-logs by relative preference feedback. The main idea of relative preferences is that if an item on position i for a specific query is clicked, it should be more relevant than all unclicked items on positions $j < i$.[1] Assuming that a user scans the website from top to bottom, these relative preference pairs account for the position bias since only examined documents are compared. However, this approach can only infer preferences that oppose the presented order, which causes learning algorithms to reverse the status quo rankings when trained with such preferences [JSS17].

Hence, more recent approaches try to directly incorporate the probability of examination of an item in the learning algorithm. The assumption of these models is that the event of a user clicking on a document is dependent on only two factors, whether the item is being observed by the user and whether the item is relevant [JSS17]. The following equation models this dependency:

$$\Pr(C = 1 \,|q, d, p) = \underbrace{\Pr(E = 1 \,|\, p)}_{\text{examination} := \beta_p} \underbrace{\Pr(R = 1 \,|\, q, d)}_{\text{relevance} := s_{qd}}, \tag{2.4}$$

with the binary random variables C, E and R indicating whether a document is clicked, examined or relevant, respectively. Thus, $\Pr(C = 1 \,|q, d, p)$ constitutes the probability of a click on document d displayed on position p for query q. $\Pr(E = 1 \,|\, p)$ specifies the position bias, i.e. the probability of the user examining a document ranked on position p, independent of the query and the document. And $\Pr(R = 1 \,|\, q, d)$ determines the probability of d being relevant for query q, which is independent of the position.

Given n displayed rankings for query q, an unbiased estimate for the relevance of d can be determined by:

$$\hat{R}(q, d) = \frac{1}{n} \sum_{i=1}^{n} \frac{c_{qd}^{(i)}}{\Pr(E = 1 \,|\, \pi_q^{(i)}(d))} \tag{2.5}$$

where $c_{qd}^{(i)}$ is a binary variable, indicating if d is clicked on in the i-th ranking of query q and $\pi_q^{(i)}(d)$ specifies the position of d in that ranking [OR20]. Joachims et al. (2017) prove that replacing the actual relevance label y in the loss function with the propensity corrected relevance estimate $\hat{R}$ leads to the unbiased empirical risk:

$$\hat{\mathcal{R}}_{\text{IPS}}(f) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{\mathcal{L}(f(\mathbf{x}_{ij}), c_{ij})}{\Pr(E = 1 \,|\, \pi_i(j))} \tag{2.6}$$

[1] Note that while we refer to products appearing closer to the top of the SERP as being "ranked higher", the (numerical) position p they appear on is actually smaller

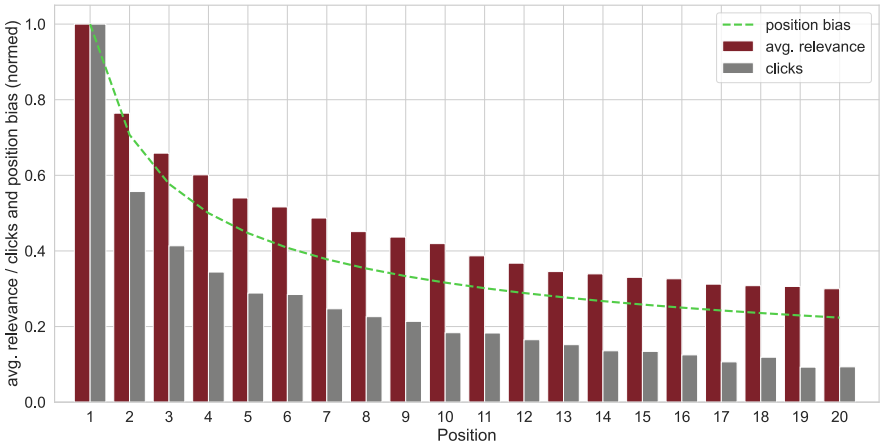


Fig. 2.3.: A comparison of the position bias with the distribution of clicks over and the average product relevance per rank position

A ranking model which is trained by minimizing the empirical risk $\hat{\mathcal{R}}_{\text{IPS}}$ is called an inverse propensity scoring (IPS) estimator and is unbiased if the click data only exhibit position bias [JSS17]. However, one problem with the IPS estimator is that neither the examination probability β_p nor the relevance s_{qd} are observed individually. Hence, the position bias also needs to be estimated from the click-logs individually.

A straightforward approach is to estimate the examination probability as the click-through rate (CTR) in position p as proposed in the *clicks over expected clicks* (COEC) method [CZ09]. This is done by dividing the sum of all clicks (over all queries and sessions) on position p by the total number of impressions on the same rank [ZJ07].

Unfortunately, the COEC model does not provide an unbiased estimate of β_p because the position-dependent CTR does not only capture the position bias, but also the rank-specific relevance. Typically, any production ranker will on average rank more relevant documents higher in the SERP, which naturally leads to fewer clicks on lower ranked documents since they are less relevant [CZ09]. As a result, normalizing by the expected CTR per position overestimates the position bias as figure 2.3 demonstrates on simulated data. One can see that the average relevance of products degrades the further down they appear in the SERP. As a consequence, not only the examination probability decreases for lower-ranked documents, but also their probability of being relevant. As a consequence, the number of clicks per position (gray bars) decreases faster than the examination probability (green line).

To tackle this issue, Dupret and Piwowarski (2008) propose to estimate β_p along with s_{qd} using the Expectation-Maximization (EM) algorithm. The EM algorithm iteratively updates s_{qd} and β_p in a way that their estimates maximize the likelihood of observing the click data c.

Lastly, several authors propose to model the position bias explicitly in the ranking function by incorporating the position as a feature with a learnable weight [CZ09; Hal20]. Though this approach helps in learning an unbiased ranking model, this ranking model depends on positional information, which is not available during online

inference. Some authors propose to set the position feature to a default value for all documents when making predictions [Lin17; Hal19], but this might lead to a considerable loss of discriminative power and unreliable inference results.

2.4 Ranking in e-commerce

In the above formulation of equation (2.5), the clicks normalized by the examination probability of a product are taken as a proxy for its relevance label. A Learning-to-Rank model can take these labels to learn optimal orderings for new queries. This approach works well in a scenario where the ranking should optimize the click-through rate (CTR). However, in e-commerce one typically aims to optimize the conversion rate (CvR) or the gross merchandise value (GMV) [Wu18a]. While the CTR depends on clicks, the CvR and GMV are measured by means of product sales.

Nonetheless, most of the existing counterfactual Learning-to-Rank methods focus on a web-search scenario where purchases as additional relevance signals do not exist. However, ignoring clicks or purchases as relevance signals can lead to a biased conception of a product's relevance. That is due to the mixed intents, which users of online shops have [Moe03]. While some users wish to purchase a specific product right away, others just want to get inspired without having an intent to purchase. For the latter kind, interesting or funny products like luxury goods might receive a lot of intention in the form of clicks. Haldar et al. (2019) observed this behavior when training an LTR model for the Airbnb search only on views, which resulted in expensive luxury properties being listed on the top of the SERP, which most users would not book.

On the other hand, simply replacing the click with the purchase signal not only reduces the amount of training data significantly, which is especially problematic for training large neural ranking models. It also suffers from the problem that cheap products are ordered more often than their often more relevant but also more expensive counterparts. Sorokina and Cantu-Paz (2016) demonstrated this problem on the example of the query "iPhone" for which a ranking model based only on order signals would rank cheap complementary products like phone cases higher than the actual phone.

This price gap between clicked and ordered items can also be observed in the Otto online shop, as figure 2.4 depicts. The plot shows the price distributions of 200,000 randomly sampled clicked (red) and purchased (green) products, where a clear shift towards more expensive products for clicks is visible.

To overcome these issues and satisfy the diverse intents of e-commerce users, multiple relevance signals should be taken into account. However, there is not much literature on learning ranking models from multiple relevance signals or in the realm of e-commerce in general, as also noted by Santu et al. (2017).

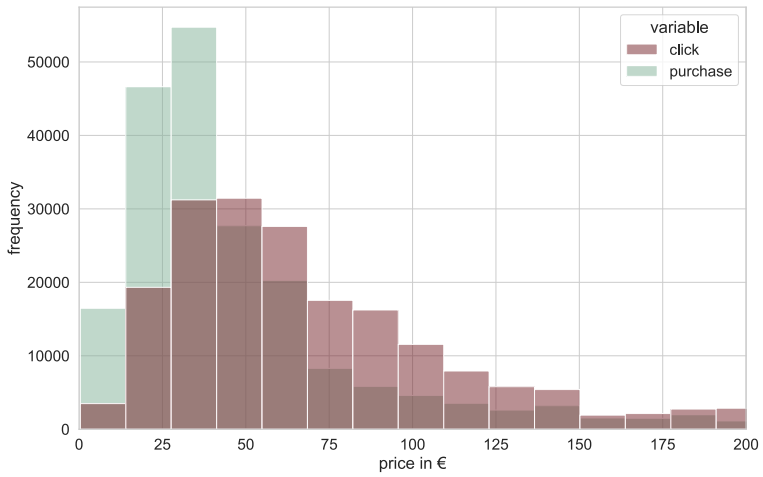


Fig. 2.4.: Distribution of the price of clicked (red) and purchased (green) items

One approach is proposed by Wu et al. (2018), who decompose the LTR problem in a click and a purchase prediction task. The authors assume that the following condition holds:

$$\Pr(O = 1|q, d) = \underbrace{\Pr(C = 1|q, d)}_{\text{click model}} \underbrace{\Pr(O = 1|C = 1, q, d)}_{\text{purchase model}}, \tag{2.7}$$

where O and C are binary random variables, indicating a purchase of and a click on document d, respectively. The authors model the click probability $\Pr(C = 1|q, d)$ by a typical ranking function with a listwise loss. On the other hand, the authors argue that the decision to purchase is made after reading the product description page and not while scanning the SERP. Consequently, they estimate the probability $\Pr(O = 1|C = 1, q, d)$ of the item being purchased after it was clicked by a binary classifier and not a ranking function. Given the probability $\Pr(O = 1|d_{ij}, q_i)$ of an item being purchased, the authors aim to optimize the expected revenue:

$$\overline{\text{GMV}} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \text{price}_{d_{ij}} \cdot \Pr(O = 1|d_{ij}, q_i)\,. \tag{2.8}$$

Another approach is proposed by Nigam et al. (2019), who classify training instances as either purchased, clicked but not purchased and not clicked. They use a pointwise 3-part hinge loss to learn to classify a document for a given query in one of these classes. The authors empirically derived threshold values for each class and the 3-part hinge loss pushes the score of a document towards the threshold value of the respective class.

2.5 Contribution

In the literature, the vast majority of existing methods have investigated unbiased Learning-to-Rank models for the web search scenario. However, as pointed out, such

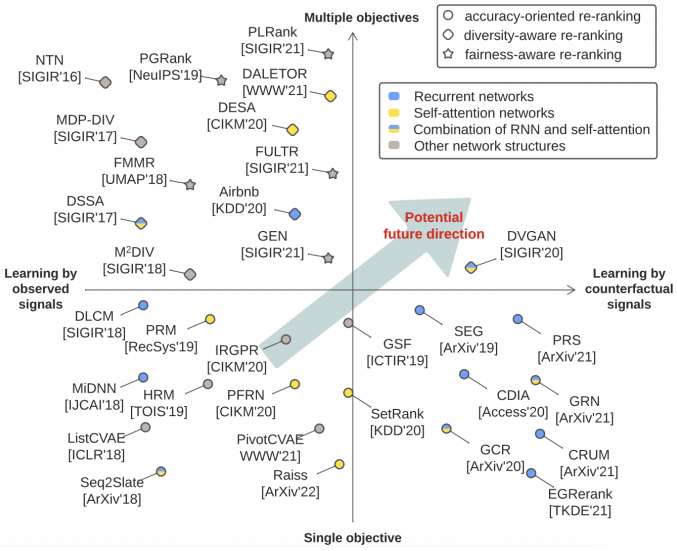


Fig. 2.5.: Visualization of the Research Gap (taken from [Liu22])

methods cannot be directly adopted by e-commerce shops since the actual relevance is measured differently.

For this reason, Nigam et al. (2019) present a neural network for the ranking task that incorporates both clicks and orders as relevance labels in a three-part hinge loss but does not consider the position bias underlying the signals. On the other hand, Ai et al. (2018) present an unbiased learning framework for neural networks but do not incorporate additional feedback. To the best of the author's knowledge, no work yet deals with neural ranking models which account for both position bias and multiple relevance signals. In a recent review of the neural ranking literature, Liu et al. (2022) found a similar research gap. The authors analyzed the current research on neural ranking regarding their objectives (single accuracy objective or multiple objectives) and the supervision signals (observed signals or counterfactual signals). Their findings are depicted in figure 2.5.

Consequently, the contributions of this thesis can be summarized as follows:

1. incorporate additional user interactions like purchases in a relevance model
2. eliminate the position bias when learning the neural ranker from customer signals
3. test the proposed method in a live experiment on a large e-commerce platform and compare the results to a strong benchmark

Therefore, this thesis will jointly concern the topics of neural ranking, counterfactual LTR and e-commerce to leverage their benefits like automatic feature generation, large user-centric training data and multiple feedback signals. To the best of the author's knowledge, this work is the first to simultaneously consider all of these areas in the development of an LTR model.

3 Development of the Unbiased Neural Ranking Model

> *All models are wrong, but some are useful*
>
> — **George Box**
> (Statistician)

In this section, a detailed description of the unbiased neural ranking model that has been developed throughout this project is given. Drawing on the findings from chapter 2, three major components have been identified that need to be incorporated in such a ranking model. First, since the aim of applying neural networks to the LTR task is to leverage raw text data, an encoder ϕ that maps search queries and product descriptions in a mutual space is required. Second, a scoring function f must be developed that maps the query-document features to a scalar relevance score to be sorted over. And lastly, a learning algorithm that incorporates multiple sources of implicit customer feedback while accounting for the position bias is required. The remainder of this chapter is structured in accordance with these identified model components.

3.1 Encoder

The first task of the neural ranking model is to determine the degree of similarity between the user query and the retrieved products. In LTR, the query is usually the most important piece of information to infer the intent of the customer and hence this part is inevitably important. Yet, compared to tasks like passage or answer re-ranking, queries and product descriptions in the realm of e-commerce are much shorter and often lack any semantics [Li21]. Hence, given their computational burden, very large and complex transformer-based models like BERT, which constitutes the state-of-the-art in passage re-ranking, do not appear to be necessary for this task. Moreover, Sarvi et al. (2020) even found pre-trained BERT models to perform worse on the product search task compared to other architectures presented in chapter 2.2. They partly attribute this to the fact that the text BERT is pre-trained on is very different from the text found in product descriptions.

Therefore, this work will start with a detailed description of the DSSM model introduced in chapter 2.2, which still enjoys great popularity in the literature [Li21; VRK16], and gradually replace parts of it with more sophisticated modeling techniques. Chapter 4.2.1 will then compare the performance of the presented model with the original DSSM and also an encoder based on the transformer architecture.

3.1.1 Tokenization

The first part of the DSSM and any language model is tokenization. Before the query and document strings can be used as input for a neural network, they need to be transformed to a numerical representation. Typically, a first step – after some text processing like lowercasing and removal of special characters – is to split the text sequence on white spaces. Each query and document is then represented as a sequence of words (tokens) $\mathbf{t} = \{t_i\}_{i=1}^{m}$, each of which can be represented as a one-hot encoded (ohe) vector:[1]

$$\mathbf{ohe}_t = \left[\mathbb{1}_{[t=t_i]}\right]_{i=1}^{|T|} \tag{3.1}$$

where each dimension corresponds to a specific token in the vocabulary T and the indicator function evaluates to 1 if t is equal to that token. Following this naive approach would lead to an input space, whose dimension is equal to the number of distinct words in the text collection. Since the number of words for a large document collection is typically several hundreds of thousands, the resulting input space would be too large to fit on standard computing hardware. Moreover, following Zipf's law, most of those tokens occur very rarely and thus receive very few updates during training, leading to bad representations of the corresponding tokens [Pia14].

Therefore, Huang et al. (2013) use letter n-grams to reduce the dimensionality of the input as well as the risk of out-of-vocabulary (OOV) tokens, i.e. tokens that cannot be mapped to a dimension in the input space since they did not appear during training. Letter n-grams are defined as all possible subsequences of n consecutive letters of a given word. Huang et al. (2013) propose trigrams, so for example "iphone" will be represented as iph, pho, hon, one. Since the number of possible trigrams for a Latin alphabet is limited, this technique leads to a significant dimensionality reduction compared to the bag-of-words vector. However, letter n-grams also come with downsides. First, letter grams lose a lot of information over the original tokens. Moreover, there exists a considerable risk of hash collisions, i.e. two words getting hashed to the same vector representation. Both problems can be mitigated by choosing a larger letter gram size, however this comes at the cost of a drastic increase in the vocabulary size.

To overcome these issues without increasing the size of the vocabulary, more modern language models use subword tokenizer. Subword tokenization is a class of tokenization strategies, that all use statistical heuristics to build the vocabulary T. This work uses the wordpiece tokenization, which is utilized in the BERT model [Dev19]. This method segments text such that common words remain as they are and less frequent words get split into subwords. The goal of the wordpiece tokenizer is to utilize as few word splits as possible while maintaining a vocabulary size $|T|$ close to some fixed value. Therefore, sub-units of tokens are included in the vocabulary based on their likelihood of occurring in the given text collection $\mathcal{W}$ [Wu16].

[1] unless stated otherwise, vectors are column vectors

Algorithm 1: Learning a Vocabulary T with the WordPiece Algorithm

```
1  Function WordPiece
2  Input word-count pairs from text collection 𝒟 = {(w, c_w)|w ∈ 𝒲}, thresh.
     H
3  T ← ∅,   𝒮 ← ∅
4  for (w, c) ∈ 𝒟 do
5  |  generate all possible wordpieces 𝒮_w of word w
6  |  for t ∈ 𝒮_w do
7  |  |  if t ∈ 𝒮 then
8  |  |  |  c_t = c_t + c
9  |  |  else
10 |  |  |  c_t = c ;   𝒮 ← 𝒮 ∪ t
11 |  |  end
12 |  end
13 end
14 sort 𝒮 in descending order of token length |t|
15 for t ∈ 𝒮 do
16 |  if c_t ≥ H then
17 |  |  T = T ∪ t
18 |  |  c_s = c_s − c_t      ∀s ∈ 𝒮_t \ t
19 |  end
20 end
21 return T
```

To do so, the wordpiece tokenization, as implemented by Google (2021), starts with counting the number of occurrences for each word w in the text collection $\mathcal{W}$ after lowercasing and whitespace splitting.The algorithm then iterates over all words and generates for each word w all possible substrings $\mathcal{S}_w$, so called wordpieces (lines 4–5). For example, the substrings for the word "hello" are h, he, hel, hell, hello, #e, #el, #ell, #ello, #l, #ll, #llo, #lo and #o, where a leading # indicates a suffix. Each wordpiece t is added to the set of all known wordpieces $\mathcal{S}$ and is given a counter c_t, which each time the wordpiece is generated by a word, is incremented by the frequency value of that word (lines 6–12). Once the count values for all wordpieces are collected, the algorithm iterates over each substring in the collection in decreasing order of their length $|t|$ (lines 14–15). A substring is kept, if the number of its counts exceeds a given threshold value H (lines 16–17). By choosing H appropriately, the size of the resulting vocabulary can be reduced to the desired size.

If a wordpiece t is added to the vocabulary, the likelihood of its sub-units $\mathcal{S}_t \setminus t$ to be used decreases proportionally to c_t. That is, once a token is added to the vocabulary, it will never be segmented further into any of its wordpieces. Since the general idea of this tokenization strategy is to only add such wordpieces to the vocabulary, that also have a high likelihood of being used, the count value of t must be subtracted from the counter of all its sub-units (line 18). By iterating through $\mathcal{S}$ in decreasing order of $|t|$, one makes sure that c_t does not change anymore after t has been added to the vocabulary [Goo22b].

After the vocabulary has been built and returned (line 21), new text can be tokenized. To do so, each word is looked up in the vocabulary T, and if it is present, it remains as it is. If it is not, the word is iteratively truncated by one character at the end until a matching substring is found in the vocabulary. For example, if hello is not present, the vocabulary will be searched for hell, hel and so on. If for instance, hel is the first match in the vocabulary, the missing suffix #lo is searched for in the vocab, and if it is not present, the same procedure repeats until the token can be fully described by wordpieces [Goo21].

This method brings several advantages. First, very important and frequently occurring words remain unchanged and hence no information loss happens. Moreover, the frequency-based approach ensures that no rare and irrelevant terms are added to the vocabulary and thus small vocabulary sizes can be accomplished without losing too much information [Kud18]. Lastly, since every single character of the alphabet should appear in the vocabulary, there are no out-of-vocabulary tokens since each word can at least be built by concatenating all its letters [Son21].

3.1.2 Encoder Architecture

Given tokens $\mathbf{t}_q$ and $\mathbf{t}_d$ for query q and document d, respectively, the input space for the encoder network can be defined by the one-hot encoded vectors $\mathbf{ohe}_t$ as described above. Token embeddings of size ϵ are generated by multiplying the one-hot vectors with a randomly initialized embedding matrix $W_E \in \mathbb{R}^{|T| \times \epsilon}$, which is learned throughout the training process. Practically, the multiplication of the sparse one-hot vectors with the embedding matrix is inefficient. Instead, the hashing trick is applied. Each term is hashed to an integer corresponding to the jth entry in the embedding matrix. Each token embedding $\mathbf{e}_t$ is then extracted from W_E by a lookup operation, which is more efficient than the multiplication [Cho15].

After the embedding layer, q and d are represented as sequences of token embeddings E_q and E_d of dimensionality $m \times \epsilon$, where m is the number of tokens the respective text is split into. In order to map variable length text sequences in a mutual space, DSSM uses a pooling operation, which generates a fixed length embedding from an arbitrary input sequence. Typically, a mean pooling operation is used here, where the average over the embeddings of all tokens is determined. The fixed length embedding is then used as input to a Multi-layer Perceptron (MLP) to provide more flexibility. Equation (3.2) summarizes the Deep Semantic Matching Model as described by Huang et al. (2013):

$$
\begin{aligned}
\mathbf{e}_{t_i} &= {W_E}^{\top} \mathbf{ohe}_{t_i}\,, \qquad i = 1, ..., m && \text{(Embedding Layer)} \\
\mathbf{z}^{(0)} &= \frac{1}{m} \sum_{i=1}^{m} \mathbf{e}_{t_i}\,, && \text{(Mean Pooling)} \\
\mathbf{z}^{(l)} &= g(W^{(l)\top} \mathbf{z}^{(l-1)} + \mathbf{b}^{(l)})\,, \qquad l = 1, ..., L && \text{(FC Layer)}
\end{aligned} \tag{3.2}
$$

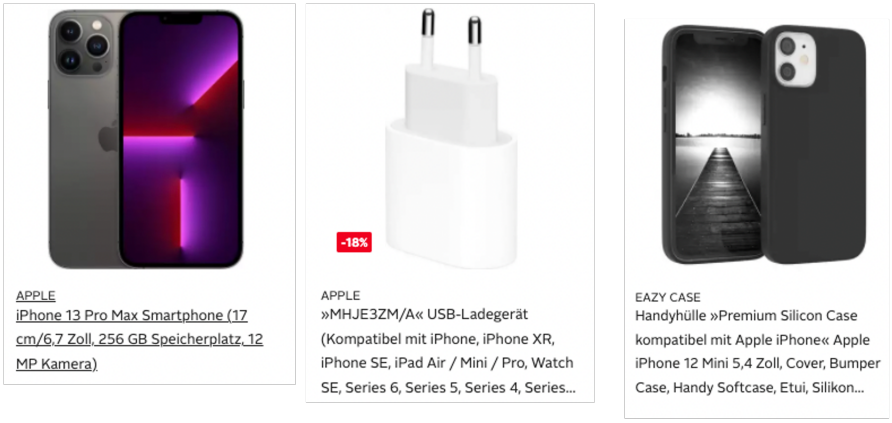


Fig. 3.1.: Different products containing the word "iPhone"

where g is a nonlinear activation function, typically a $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ or a $\text{ReLU}(x) = \max(0, x)$. Moreover, $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the learnable parameters of the MLP.

One issue with the original DSSM is the loss of word order information through the pooling operation. Previously, we argued that text sequences in e-commerce do not exhibit complex grammatical structures but rather are simple stacks of keywords. While this is essentially true, those keywords might have different meanings given the context or position they appear in. Take for example the products shown in figure 3.1. While the word "iPhone" appears in all product titles, it only refers to the product itself in the first example. In contrast, for the phone case and charging cable, it only signals that these products are compatible with the iPhone.

Recurrent Neural Networks (RNNs) are generally able to capture such dependencies. RNNs map each token to a hidden state $\mathbf{h}_{t_i}$ of dimension η, which constitutes a weighted sum of the embedding of the respective token and the hidden state $\mathbf{h}_{t_{i-1}}$ of the previous token:

$$\mathbf{h}_{t_i} = g(W_h^\top \mathbf{h}_{t_{i-1}} + W_e^\top \mathbf{e}_{t_i} + \mathbf{b}), \qquad i = 1, ..., m \tag{3.3}$$

where $W_h \in \mathbb{R}^{\eta \times \eta}$ and $W_e \in \mathbb{R}^{\epsilon \times \eta}$ are learnable weight matrices which are shared across all tokens, $\mathbf{b}$ is a bias term and g constitutes a nonlinear activation function [GBC16]. The elements of the initial hidden state $\mathbf{h}_{t_0}$ are typically set to zero. The idea of RNNs is that they incorporate information of all previous tokens when mapping $\mathbf{e}_{t_i}$ to a hidden space. Thus, the network would retain the information it got from the keyword "case" when encoding the token "iPhone". Consequently, the hidden states are not fixed token embeddings but change upon the context they appear in. Thus, the hidden state of the token "iPhone" in the product description of the case may be very different from that displayed in the title of the iPhone itself. This makes RNNs a powerful architecture for learning product representations, as they have enough flexibility to even map products with textual overlap far apart in embedding space.

The issue with vanilla RNNs as in equation (3.3) is that they are only able to retain information from nearby tokens but not from tokens that appear further back in the sequence. This stems from the fact that the token information is propagated through the whole sequence and each hidden state loses parts of the information from the previous token. To overcome this issue, this work uses Gated Recurrent Units (GRUs), which decide for each token in the sequence how much of its information should be retained in the hidden state $\mathbf{h}$ [Cho14]. They do so by employing a forget gate mechanism Γ, defined as:

$$\Gamma_{t_i} = \sigma(W_h^{\Gamma\top}\mathbf{h}_{t_{i-1}} + W_e^{\Gamma\top}\mathbf{e}_{t_i})\,, \qquad i = 1, ..., m \tag{3.4}$$

with the sigmoid activation function σ squishing the output to the range $[0, 1]$. The forget gate then uses these weights to determine how much the current token t_i influences the hidden state by using the following update equation:

$$\mathbf{h}_{t_i} = (1 - \Gamma_{t_i}) \odot \mathbf{h}_{t_{i-1}} + \Gamma_{t_i} \odot \tilde{\mathbf{h}}_{t_i}\,, \qquad i = 1, ..., m \tag{3.5}$$

where $\tilde{\mathbf{h}}_{t_i}$ is the candidate hidden state, which is similar to the formulation of the vanilla RNN, i.e.:

$$\tilde{\mathbf{h}}_{t_i} = \tanh(W_h^\top(\Gamma_{t_i} \odot \mathbf{h}_{t_{i-1}}) + W_e^\top\,\mathbf{e}_{t_i} + \mathbf{b})\,, \qquad i = 1, ..., m \tag{3.6}$$

Note that for simplicity equations (3.4–3.6) define the Minimal Gated Unit proposed by Zhou et al. (2016). However, the implementation this work uses is based on the original GRU developed by Cho et al. (2014).

Another shortcoming of the DSSM is that the pooling operation treats each token embedding as equally important. However, product descriptions typically contain useless information like stopwords and especially for longer text descriptions, these useless tokens can dilute the document embedding when simply taking the average. Therefore, we apply an attention mechanism, similar to [Wu16]. The attention mechanism learns to detect the important tokens in a sequence and gives a larger weight to them. In the end, a weighted average is applied over all tokens to combine the individual token representations into a single document embedding. To generate the weights, the hidden states produced by the GRU cell are passed through a layer of fully connected units and a single output node per token. A softmax activation is applied to the result in order to get a discrete probability distribution over tokens. These are the token-specific weights that can be learned by the network through the linear projection matrix $W_g \in \mathbb{R}^{\eta\times 1}$. Formally, the operation can be described by:

$$g_{t_i} = \frac{\exp(W_g^\top \mathbf{h}_{t_i})}{\sum_{j=1}^{m} \exp(W_g^\top \mathbf{h}_{t_j})}\,, \qquad i = 1, ..., m \tag{3.7}$$

The result is again passed through a Multi-layer Perceptron to produce the final embedding $\mathbf{z}^{(L)}$. Equation (3.8) summarizes this model, which will be referred to as GRU-DSSM:

$$\begin{aligned}
\mathbf{e}_{t_i} &= W_E^\top \mathbf{ohe}_{t_i}\,, && i = 1, ..., m && \text{(Embedding Layer)} \\
\mathbf{h}_{t_i} &= \mathrm{GRU}(\mathbf{h}_{t_{i-1}}, \mathbf{e}_{t_i}; \theta)\,, && i = 1, ..., m && \text{(GRU Layer)} \\
g_{t_i} &= \frac{\exp(W_g^\top \mathbf{h}_{t_i})}{\sum_{j=1}^{m} \exp(W_g^\top \mathbf{h}_{t_j})}\,, && i = 1, ..., m && \text{(Attention Gate)} \\
\mathbf{z}^{(0)} &= \sum_{i=1}^{m} g_{t_i}^\top \mathbf{h}_{t_i}\,, && && \text{(Weighted Average)} \\
\mathbf{z}^{(l)} &= g(\mathbf{W}^{(l)\top} \mathbf{z}^{(l-1)} + \mathbf{b}^{(l)})\,, && l = 1, ..., L && \text{(FC Layer)}
\end{aligned} \tag{3.8}$$

where $\mathrm{GRU}(\cdot)$ refers to the GRU cell described by equations (3.4–3.6) and θ includes all learnable parameters of this layer. Lastly, let $\mathbf{v}$ denote the ℓ^2-normalized document embedding $\mathbf{z}^{(L)}$, so that the cosine similarity between query q and document d can be written as $\mathbf{v}_q^\top \mathbf{v}_d$.

3.1.3 Encoder Training

To learn the parameters of the encoder, a contrastive learning strategy is employed. The goal of contrastive representation learning is to learn an embedding space such that similar sample pairs stay close to each other while dissimilar ones are far apart. The approach was originally developed in the popular FaceNet paper [SKP15], which uses a triplet loss:

$$\mathcal{L} = \max\left(0, \rho + \xi\left(\mathbf{v}_q, \mathbf{v}_{d^+}\right) - \xi\left(\mathbf{v}_q, \mathbf{v}_{d^-}\right)\right)\,, \tag{3.9}$$

where ρ is the margin parameter and ξ is the Euclidean distance between anchor and document embeddings. This loss is non-zero if the negative example d^- is not mapped at least ρ distance units farther from the anchor in Euclidean space than the positive example. Thus, the triplet loss ensures that non-matching examples d^- are pushed away from the anchor q while matching examples d^+ are dragged towards it. FaceNet uses labeled data consisting of photos from thousands of people, each photo labeled with an ID of the person the photo is showing. Given a photo of a person (the anchor), a different photo from the same person is used as a positive example d^+ and a photo of a different person as a negative example d^-. While this approach works well in a scenario, where anchors, as well as positive and negative items, are coming from the same domain, it is not feasible in this work where anchors are user-issued queries and examples are product descriptions.

Therefore, this work uses click-logs in order to specify relevant products for a given query. If product d has been clicked on for query q, it constitutes a positive

example. While it is straightforward to define positive examples with click-logs, it is much more difficult to gather negative ones. Robinson et al. (2021) define two criteria that make a good negative sampling strategy. First, the sampling distribution should only sample true negatives, i.e. documents that are indeed not relevant to the query. And second, it should be able to generate negative samples that are difficult for the current embedding to discriminate. Using unclicked products from the SERP would result in samples that are difficult to distinguish as negatives since the retrieval system only returns candidates that are similar to the query to some degree. However, this sampling strategy would introduce too many false negatives, especially in e-commerce applications, where often hundreds of similar products are retrieved. In such cases, the absence of a click does not mean that the product is irrelevant but can be attributed to a lot of factors, like user preferences or position bias.

Another approach would be to sample d_i^- uniformly from the entire product catalog. While this approach would lead to very few false negatives, it does not generate sufficient hard to classify examples either. Already after a few training steps, the positive examples will be much closer to the anchor than most of the negatives, resulting in vanishing gradients and premature convergence of the model.

A popular sampling strategy that ensures hard negatives while also reducing the risk of false negatives is the in-batch sampling strategy proposed by [Wu18b]. It utilizes the common practice in deep learning to train a neural network on mini-batches instead of a single training example at a time. Given a dataset $\mathcal{D} = \{q_i, d_i^+\}_{i=1}^n$, which consists of anchors q_i and corresponding positive items d_i^+, in each training step N^B training examples are randomly sampled from $\mathcal{D}$ without replacement and put in a mini-batch B. Queries and products in B are then passed through the encoder network to produce their embeddings. Then, for a given query-product pair in B, the cosine similarities of all other products in the batch to the query are determined and the product whose embedding is the closest to the query is used as a negative example for it:

$$d_i^- = \underset{d_j \in B \setminus d_i^+}{\operatorname{argmax}} (\mathbf{v}_{q_i}^\top \mathbf{v}_{d_j}) \qquad \forall i \in B \tag{3.10}$$

Even with an adequate sampling strategy, Sohn (2016) has found that representation learning with triplet loss often yields bad performance due to convergence in poor local optima. The authors argue that only large enough batch sizes and a diverse collection of negative samples make the training challenging enough for the model to learn meaningful representation so as to distinguish different examples. Therefore, in more recent work, multiple negatives for each positive example are used. Sohn (2016) for example uses the $N^N > 1$ negatives that are the closest to the anchor while Robinson et al. (2021) propose to use a documents distance as weight for sampling N^N negatives. And Chen et al. (2020) even utilize all other documents in the batch as negatives.

In order to learn the parameters of the encoder with N^N negative examples $D_i^- = \{d_{ij}^-\}_{j=1}^{N^N}$ per query q_i, the triplet loss has to be replaced with a listwise loss function. Most authors propose to use cross-entropy loss, which is a measure of the difference between two probability distributions. While the target distribution y is equal to 1 for the positive example and zero for all negatives, query-document embeddings are mapped to a probability distribution using the softmax function with the cosine similarities as inputs:

$$\Pr(d_i^+ | \mathbf{v}_{q_i}, \mathbf{v}_{d_i^+}) = \frac{\exp(\mathbf{v}_{q_i}^\top \mathbf{v}_{d_i^+})}{\exp(\mathbf{v}_{q_i}^\top \mathbf{v}_{d_i^+}) + \sum_{j=1}^{N^N} \exp(\mathbf{v}_{q_i}^\top \mathbf{v}_{d_{ij}^-})} \tag{3.11}$$

The softmax provides a discrete probability distribution where $\Pr(\cdot)$ denotes the probability of item d_i^+ correctly being classified as the positive example. The learning objective is to maximize this probability for all training examples, i.e. to maximize the joint probability $\prod_{i=1}^{n} \Pr(d_i^+ | \mathbf{v}_{q_i}, \mathbf{v}_{d_i^+})$ or equivalently to minimize the negative log-likelihood

$$\mathcal{L} = \sum_{i=1}^{n} -\log \left(\Pr(d_i^+ | \mathbf{v}_{q_i}, \mathbf{v}_{d_i^+}) \right) . \tag{3.12}$$

Equation (3.12) is called the softmax cross-entropy loss or softmax loss for short. In section 4.2.1 an overview of the performance for the different sampling strategies and loss functions is given.

A big advantage of in-batch negative sampling is that a single forward pass suffices to compare each query with every product in the mini-batch, thus not introducing any additional computational costs except for the calculation of the dot products. Note that this online mining of negative examples is only feasible for representation-based models. The representation-based embeddings can be compared after a complete forward pass through the encoder due to the late interaction property of such architectures (recall figure 2.2). For interaction-based models, the interaction happens right after the embedding layer. Thus each possible pair of samples in the batch would need a forward pass through the model, raising the complexity for in-batch mining from $\mathcal{O}(N^B)$ to $\mathcal{O}(N^{B^2})$ and making these models prohibitively expensive to train with this strategy. Consequently, Xiong et al. (2017) propose to train the DRMM from preference pairs built from relevance labels, which we do not have. Therefore, interaction-based models are not considered further as potential encoder networks.

3.1.4 Model Improvements

When training the encoder network with the softmax loss using the cosine similarities as logits, we found that the model does not converge, regardless of the batch size and the learning rate. After decreasing the loss a little bit within the first few steps, the loss would not decrease further but get stuck on a very large value. After inspecting

the model, we found that the problem was caused by the use of the cosine similarities as inputs to the softmax loss. Since the cosine similarity is bound to the range $[-1, 1]$, the probability $\Pr(d_i^+|\mathbf{v}_{q_i}, \mathbf{v}_{d_i^+})$ cannot get close to 1, even if the network perfectly discriminates positives and negatives. Suppose a batch size of $N^B = 512$ and that the cosine similarity is -1 for all negative pairs and 1 for the positive pair. In this case, the model has separated the positive and negative pairs perfectly, but the probability of item i correctly being classified as the positive example is only:

$$\Pr(d_i^+|\mathbf{v}_{q_i}, \mathbf{v}_{d_i^+}) = \frac{\exp(1)}{\exp(1) + 511 \cdot \exp(-1)} = 0.014 \tag{3.13}$$

Since the gradient of the softmax loss w.r.t. the model output is $1 - \Pr(d_i^+|\mathbf{v}_{q_i}, \mathbf{v}_{d_i^+})$, large gradients still point in directions where no further improvement can be achieved while incorrectly classified examples do not receive sufficient attention. In the end, this causes the model to not converge.

To overcome the range problem introduced by the cosine similarity, one could simply omit the vector normalization and use the plain dot product of query and document embeddings as logits. A logical consequence of the inner product is that its value not only depends on the direction the respective vectors point to, but also on their length. However, the classification decision should be determined merely by the direction [Soh16]. For example, Chen et al. (2020) found learned representations to be significantly worse when using the dot product instead of the cosine similarity. This might be due to the regularization, the ℓ^2-normalization adds to the embedding space, leading to more discriminative features. Moreover, cosine similarity is strongly preferred over the unnormalized inner product when used as a feature in downstream tasks, like the LTR model [Wan17].

To improve the training process and model convergence while also preserving the embedding direction as sole source of the classification decision, Chen et al. (2020) employ a temperature scaled softmax cross-entropy loss with ℓ^2-normalized embeddings. More specifically, the softmax activation function from equation (3.11) becomes:

$$\Pr(d_i^+|\mathbf{v}_{q_i}, \mathbf{v}_{d_i^+}) = \frac{\exp(\mathbf{v}_{q_i}^\top \mathbf{v}_{d_i^+}/\tau)}{\exp(\mathbf{v}_{q_i}^\top \mathbf{v}_{d_i^+}/\tau) + \sum_{j=1}^{N^N} \exp(\mathbf{v}_{q_i}^\top \mathbf{v}_{d_{ij}^-}/\tau)} \tag{3.14}$$

where τ is the scaling parameter, typically referred to as temperature. By choosing τ small enough, the probabilities of the softmax are calibrated, leading to convergence similar to that with unnormalized dot products while generating more meaningful embeddings. Figure 3.2 illustrates the training losses for the three different versions of the softmax (left) and the upper bound of the softmax operation for different scaling values $1/\tau$ (right).

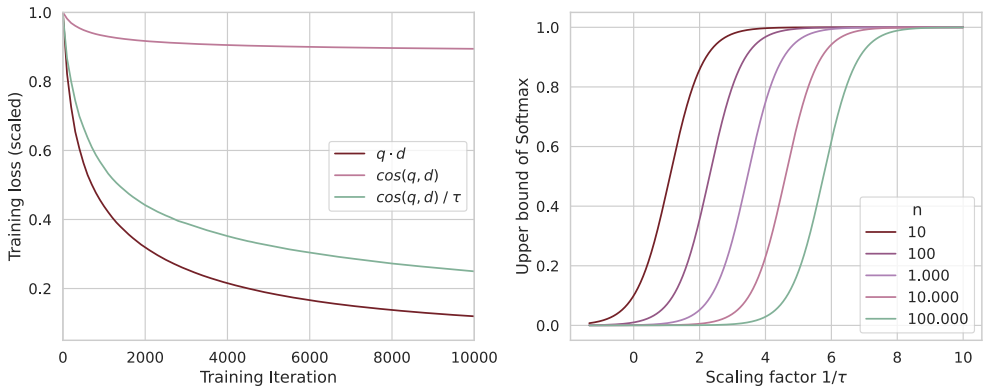


Fig. 3.2.: Left plot shows the training loss convergence for the unnormalized $(q \cdot d)$, the ℓ^2-normalized $(cos(q, d))$ and the temperature-scaled normalized $(cos(q, d)/\tau)$ version of the softmax. The right plot shows the upper bound of the temperature-scaled softmax for different scaling factors $1/\tau$

3.1.5 Product Category Classification

We now study, how the encoder ϕ can act as a feature extractor for a downstream task. Each product belongs to one of more than 1,500 classes, which we refer to as the product base class or PBK ("Produkt-Basisklasse") for short. The embeddings learned by the encoder should already encapsulate a lot of information required to specify the PBK of a product based on its description. A classifier that takes these embeddings as input can consequently learn to extract the relevant information from it to predict the PBK for a given product. This can be regarded as useful for two reasons. First, since labeled data for this task exists, it can be used to evaluate the quality of the learned embeddings and compare the different architectures presented. The idea is that the better the encoder captures and compresses the raw text, the better the corresponding classifier learns to classify the product into PBKs.

Second, we will use a similar classifier on the query embeddings in order to predict the PBK, the user probably had in mind. This prediction can then be compared against the PBK of the retrieved products and thus serve as a feature for the ranking function. Of course, there exists no true label specifying the desired PBK of a customer given his/her query. Instead, the distribution of past clicks over PBKs for a given query is determined and serves as a label for this task. Given a set $\mathcal{D}_q = \{D_q^{(i)}\}_{i=1}^n$ of n presented rankings for a query q, the label for a specific pbk is defined as follows:

$$y_{pbk}^q = \frac{\sum_{D_q \in \mathcal{D}_q} \sum_{d \in D_q} \mathbb{1}_{[pbk_d = pbk]} \cdot c_d}{\sum_{D_q \in \mathcal{D}_q} \sum_{d \in D_q} c_d}, \qquad \forall pbk \in \mathcal{C}, q \in Q \tag{3.15}$$

where $\mathbb{1}$ is the indicator function, evaluating to one if document d belongs to the class pbk, c_d is a binary variable indicating a click on document d and $\mathcal{C}$ is the set of all PBKs.

In figure 3.3 the resulting labels are displayed for two example queries. In general, this approach captures the user intent for rather clear queries quite well (a user

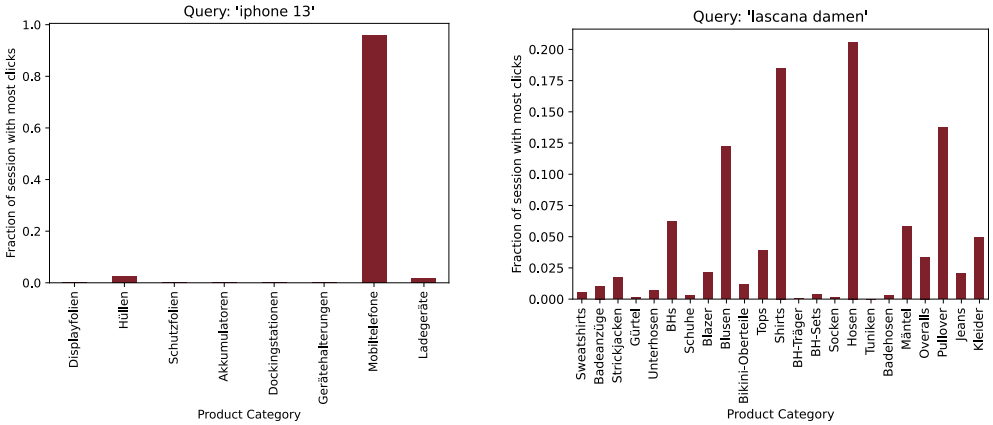


Fig. 3.3.: Distribution of most clicked product classes for example queries

searching for "iPhone" typically has a mobile phone in mind) and distributes the probability mass to several relevant product classes for queries, that cannot be associated with a single category alone (especially brand names typically belong to a set of categories).

The classification task in both scenarios is identical. The goal is to find a mapping $f(\mathbf{v}) : \mathbb{R}^{\epsilon} \rightarrow \mathbb{R}^{|\mathcal{C}|}$ from the embedding space to a probability distribution over the distinct PBKs in $\mathcal{C}$. We use a Multi-layer Perceptron with two hidden layers of fully connected units and ReLU activation functions as classifier. Moreover, batch normalization [IS15] and dropout [Sri14] are applied during training. The final layer consists of $|\mathcal{C}|$ fully connected units and again the softmax is used as an activation to map the final scores to a probability simplex, i.e.:

$$\Pr(i|\mathbf{v}) = \frac{\exp(f(\mathbf{v}))}{\sum_{j=1}^{|\mathcal{C}|} \exp(f(\mathbf{v}))} \qquad \forall i = 1, ..., |\mathcal{C}| \tag{3.16}$$

As before, these probabilities can be used together with the labels y to determine the cross-entropy loss:

$$\mathcal{L} = \sum_{i=1}^{|\mathcal{C}|} -y_i \log(\Pr(i|\mathbf{v})) \tag{3.17}$$

3.2 Scoring Function

Most existing neural ranking models focus on learning representations of the raw texts of query and document and define a scoring function, which takes these representations as input [Guo20a]. The representation-based encoder networks implement this scoring function as the cosine similarity between query and document embeddings. Interaction-based models like the DRMM use more sophisticated scoring functions, consisting of several layers of fully connected units. And MatchPyramid [Pan16] even

uses a deep convolutional network as a scoring function, which takes the matching matrix as input.

However, all these models only incorporate textual information of queries and documents in the scoring function. Thus, relevance is solely measured as the semantic similarity between query and document. This might work well in passage re-ranking or in question-answer retrieval systems. In the realm of e-commerce however, it is not unusual that hundreds of products from the product catalog are a perfect semantic match to the search query. The relevance of an item in such cases is not only dependent on the match between query and document, but also on user-specific preferences, other product attributes like its price, availability or popularity and not least on the degree of competition between the products in the SERP [SSZ17]. The matching models cannot adequately model such complex dependencies, therefore a ranking function $f(\phi(q, d), \mathbf{x}_{qd})$, using semantic matching scores from the encoder ϕ along with additional features to produce the ranking score s, is developed.

The Deep Relevance Matching Model is the first neural ranker to consider the difference between semantic matching, which is to measure the semantic similarity between two texts and relevance matching, which describes the task of ranking documents by their relevance to the user's query. The authors Guo et al. (2016) argue that while semantic matching is an important prerequisite for relevance matching, exact term matching is still more important for relevance matching and cannot be replaced by semantic matching models like DSSM.

This work follows the argumentation of the authors and views the encoder ϕ only as a prerequisite for an adequate ranking but not as a relevance matching model on its own. Especially in online shops, queries often contain very detailed descriptions like the model number of the desired product. For example, the query for a specific impact drill could be "Bosch GSB13". Here the model number "GSB13" is the most important part of the query, which probably uniquely identifies the product the customer requires. Semantic matching models will fail on such queries since rare tokens like "GSB13" get very few updates during training and are consequently poorly represented in embedding space. The DRMM, on the other hand, solves this issue by explicitly accounting for exact matches in the matching function and assigning different weights to each query token, similar to the BM25 model. However, it still focuses only on textual representations of queries and documents, thus neglecting attributes of queries and products that cannot be inferred from the raw text.

To the best of the author's knowledge, Shan et al. (2016) are the only authors to incorporate additional features such as document popularity as input to a neural ranking model. This is in line with the findings of a recent literature review of the neural ranking field by Trabelsi et al. (2021). And also, Ai et al. (2019) found that neural ranking techniques either focus on advanced representations of document and query text or on the development of proper scoring functions, in which case standard LTR features are used with simple matching models like BM25. Also, large benchmark datasets incorporate either hand-crafted features like PageRank, BM25

etc. [CC11] or raw text [Baj18], which makes research on models using a combination of both features difficult and impossible to compare with other models.

Therefore, this work follows the approach of Shan et al. (2016) and concatenates the features from the encoder network to the vector of additional hand-crafted features, which are described in table B.1. We tried different ways to combine the query and document embeddings in the feature space of the scoring function, like concatenation, subtraction or summation but found the cosine similarity $\mathbf{v}_q^\top \mathbf{v}_d$ to perform best.

In addition to popularity-based features like the rating or past sales of a product, we also use the BM25 score between query and document to account for the problem arising from the semantic comparison described above.

A lot of the hand-crafted features exhibit skewed distributions or extreme scales. For example, the number of past clicks can be on the scale of millions for some items while other items only have a few hundred or even no clicks at all. In contrast to other machine learning models like Decision Trees, Neural Networks are known to be sensitive to such patterns in the input data, which can result in less optimal models and introduce numerical instability during training [BN06]. Therefore, feature normalization is an essential step to improve the effectiveness of neural ranking models when using hand-crafted features. Since a lot of features are count data following a power-law distribution (e.g. number of clicks, number of reviews), this work uses a logarithmic transformation to normalize these features. However, count-based features can be zero for a lot of observations and since the logarithm is undefined for $x \leq 0$, this work employs the log1p-transformation $\tilde{x} = \log(1 + x)$ similar to Zhuang et al. (2020). Furthermore, to avoid varying scales in the input space, all input features are normalized using the Gaussian (or z-score) transformation similar to Agarap (2019):

$$\tilde{x} = \frac{x - \mu}{\sigma}, \tag{3.18}$$

where μ represents the mean value for each feature x and σ represents the corresponding standard deviation. The concatenation of the transformed features with the cosine similarity of the query and product embeddings, as well as the matching score of a product's PBK with the predicted user intended PBK, is then used as input for the scoring function.

As described in section 2.1 the scoring function $f(\cdot)$ maps the retrieved documents for a query to a real-valued score, based on which these documents are sorted in descending order to generate the permutation $\tilde{\pi}$. Most existing LTR models use a uni-variate scoring function $f(q, d)$ to map a single query-product pair at a time to a relevance score instead of the whole SERP D_q. Thus, the relevance score of a product to a query is computed independently of the other documents in the list. This is also true for listwise LTR models like LambdaMART, since these models take other documents in the SERP only into account when computing the loss but not while mapping documents to a relevance score [Ai18a].

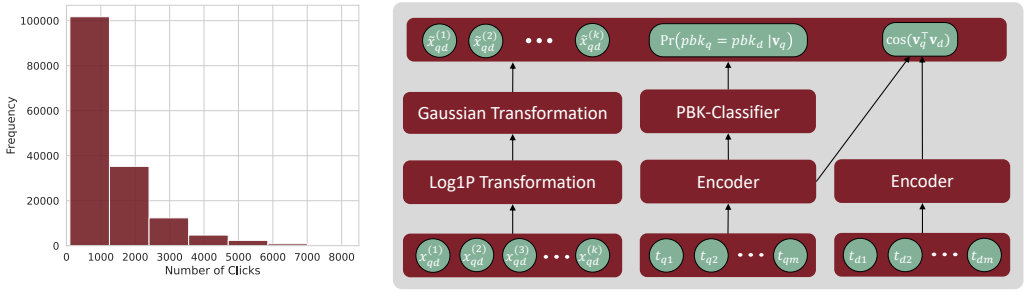


Fig. 3.4.: Distribution of the number of clicks per product (left) and illustration of the Input Space for the Neural Ranking Model (right)

Uni-variate scoring functions are consequently not able to describe cross-product interactions. However, those interactions are prevalent since each product is competing with the other products of the SERP for the user's attention. Especially in the area of e-commerce, where the customer is shown several relevant products but usually only decides on a few of them. As a consequence, a product's probability of being purchased might be much higher if the product is displayed only with less relevant items compared to a situation, where a lot of relevant items are retrieved [Bel19]. A model that ignores such dependencies might suffer from omitted-variable bias by trying to attribute these interactions to other features in the input space.

To overcome this limitation, we follow the approach described by Pasumarthi et al. (2020) and use multiple layers of Multi-Head Self-Attention (MHSA) to encode cross-document interactions. Self-attention is a mechanism that is applied to sequences of data, where a sequence in the LTR setting corresponds to the documents retrieved for a given query represented by their feature vector $\mathbf{x}$. Self-attention allows a direct comparison between the distinct items in the list by applying the scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_Q}}\right)V\,, \tag{3.19}$$

where $Q \in \mathbb{R}^{m\times d_Q}$, $K \in \mathbb{R}^{m\times d_Q}$ and $V \in \mathbb{R}^{m\times d_V}$ are different linear projections of the input $X = [\mathbf{x}_1^q, ..., \mathbf{x}_m^q]$:

$$Q = XW^Q\,, \qquad K = XW^K\,, \qquad XW^V \tag{3.20}$$

with linear projection matrices W^Q, W^K and W^V [Vas17]. The dot-product $QK \in \mathbb{R}^{m\times m}$ generates a pairwise similarity matrix of the linearly projected documents, which is scaled by the square root of d_Q to avoid numerical overflow. Based on these pairwise similarity scores, the softmax function in (3.19) computes the attention weights:

$$attn_{ij} = \frac{\exp(Q_{i:}K_{:j}^\top/\sqrt{d_Q})}{\exp(\sum_{p=1}^m Q_{i:}K_{:p}^\top/\sqrt{d_Q})} \qquad \forall i, j = 1, ..., m \tag{3.21}$$

which essentially describe, how much product d_j influences the hidden state SA_i of product d_i. The hidden state of product d_i is then determined by a weighted sum over the rows of the third linear projection $V \in \mathbb{R}^{m \times d_V}$, i.e.:

$$SA_{i:} = \sum_{j=1}^{m} attn_{ij} \cdot V_{j:} \qquad \forall i = 1, ..., m \tag{3.22}$$

Each row i $(1, ..., m)$ of the resulting matrix $SA \in \mathbb{R}^{m \times d_V}$ can be interpreted as a context vector for product d_i incorporating information of all products in the SERP including d_i itself. For example, one dimension of the context vector might determine the price of the product relative to all other products, while another indicates the average similarity score in the list. The attention scores $attn_{ij}$ determine how much each product influences the values in the context vector of another product. For example, the network might focus on the product with the highest price in the list, the products that are the most similar to the target product or only on the target product itself. Thus, complex cross-product interactions can be encoded in the context vectors.

To give the model more flexibility and allow it to attend to different parts of the SERP, Vaswani et al. (2017) propose to project the input h times with different projection matrices and apply the attention function of equation (3.19) to each of these. The outputs of the h self-attention operations are concatenated and mapped through a linear projection layer $W^O \in \mathbb{R}^{hd_V \times d_{\text{model}}}$ to form the output of the Multi-Head Self-Attention:

$$\text{MHSA}(X) = \text{concat}\left(\left[\text{Attention}(XW_i^Q, XW_i^K, XW_i^V)\right]_{i=1}^{h}\right) W^O\,, \tag{3.23}$$

where this work follows the convention to set $d_Q = d_V = d_{\text{model}}/h$ [Vas17]. Finally, inspired by residual connections in ResNet [He15], the input matrix X is added to the output of the MHSA layer and layer normalization, as proposed by Ba et al. (2016), is applied to avoid numerical overflow. For the residual connection to work, X is mapped to a d_{model}-dimensional space using a linear projection matrix $W^X \in \mathbb{R}^{k \times d_{\text{model}}}$ prior of being used as input to the MHSA layer. To further improve the expressiveness of the context vectors, the MHSA layer is repeated L^{SA}-times, with the input to subsequent MHSA layers being the output of the immediately preceding layer.

Pasumarthi et al. (2020) then combine the output of the stack of MHSA-layers with the original feature vector through concatenation and learn a feed-forward network on top of that to produce the final ranking scores. However, Beutel et al. (2018) show that a concatenation of context and feature embeddings leads to inferior results and propose the latent cross, an element-wise multiplication of the form:

$$\tilde{\mathbf{x}}_i = (1 + \mathbf{a}_i) \odot \mathbf{x}_i\,, \tag{3.24}$$

where $\mathbf{a}_i$ and $\mathbf{x}_i$ are the context embedding and the feature representation for product d_i respectively. Similar to Pasumarthi et al. (2020), we pass the resulting context-

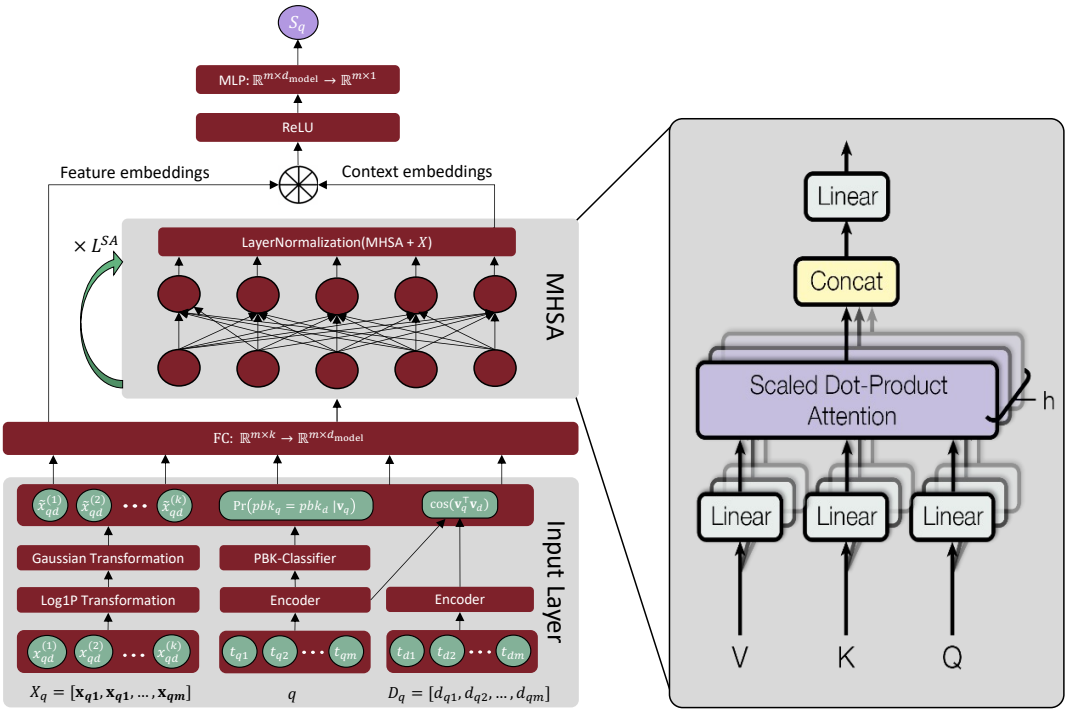


Fig. 3.5.: Illustration of the architecture of the Scoring Function f and a detailed view on the Multi-Head Self-Attention layer. The latter is taken from [Vas17]

aware embedding $\tilde{\mathbf{x}}_i$ of product d_i through a Multi-layer Perceptron with a single output node in the last hidden layer to produce the final ranking score s_i. Figure 3.5 illustrates the architecture of the proposed scoring function, which we will refer to as AttnRank. AttnRank is also summarized in the following set of equations:

$$
\begin{aligned}
X^{(0)} &= XW^X && && \text{(Input Projection)} \\
X^{(l)} &= \text{LN}(\text{MHSA}(X^{(l-1)}) + X^{(l-1)}), && \forall l = 1, ..., L^{\text{SA}} && \text{(SA Layers)} \\
\tilde{X}^{(0)} &= \sigma(X^{(0)} \circledast X^{(L^{\text{SA}})}) && && \text{(Latent Cross)} \\
\tilde{X}^{(\tilde{l})} &= \sigma(\tilde{X}^{(\tilde{l}-1)} W^{(\tilde{l})} + \mathbf{b}^{(\tilde{l})}), && \forall \tilde{l} = 1, ..., L-1 && \text{(FC Layers)} \\
S &= \tilde{X}^{(L-1)} W^L + \mathbf{b}^L && && \text{(Score)}
\end{aligned} \tag{3.25}
$$

where σ is the ReLU activation function, $\text{LN}(\cdot)$ refers to layer normalization and $\circledast$ denotes the latent cross operation, i.e. $A \circledast B \equiv (1 + B) \odot A$.

3.3 Parameter Learning

In order to train the ranking function, this work uses implicit user feedback gathered from click-logs. More specifically, the click-log data consists of n distinct query sessions q, where we define a query session as a specific query issued within a specific user session. There exists an N:N relationship between queries and user sessions, i.e. a user can issue several queries within a session and a query can occur in multiple

sessions. Hence, by referring to q as query session, the same search term (which we also refer to as q for ease of notation) may appear multiple times in the dataset. For each query session q, there exists a corresponding set of retrieved products:

$$D_q = \left(X_q = \{\mathbf{x}_{qd}\}, P_q = \{\mathbf{p}_{qd}\}, \mathbf{y}_q^c = \{y_{qd}^c\}, \mathbf{y}_q^o = \{y_{qd}^o\}\right)_{d=1}^{m_q} \tag{3.26}$$

where $\mathbf{x}_{qd}$ and $\mathbf{p}_{qd}$ are the feature vector and the position of product d for query q, respectively. Moreover, y^c and y^o are the binary relevance labels indicating whether a product was clicked on or purchased, respectively. The full dataset of all query sessions is denoted as $\mathcal{D} = \{D_{q_i}\}_{i=1}^n$.

Note that the parameters of the encoder model are frozen when training the scoring function, hence $\mathbf{x}$ describes the full feature set with the semantic similarity score and the PBK prediction concatenated to the hand-crafted features. This work refrains from fine-tuning the encoder on the ranking task because, as argued earlier, relevance matching is very different from semantic matching, especially in e-commerce. Thus, fine-tuning the encoder might cause it to lose the learned semantic similarity information.

3.3.1 Loss Function

As described in chapter 2.1, the loss function is a crucial part of LTR models and the requirements for an appropriate loss function are what distinguish LTR from other supervised learning tasks. A good loss function should correlate with the ranking metrics used to evaluate the ranking model, but which are themselves non-differentiable. There are two families of ranking metrics. The first are used with binary relevance labels to evaluate the quality of the permutation $\tilde{\pi}$. The most widely used binary ranking metrics are the Mean Reciprocal Rank (MRR) and the Mean Average Precision (MAP).

The MRR is a simple evaluation metric, which determines the reciprocal of the rank position of the first relevant (i.e. clicked or ordered) document in the permutation $\tilde{\pi}$. Formally, the reciprocal rank for a query q and associated documents D_q is defined as:

$$\text{RR} = \max_{d \in D_q} \frac{y_d}{\tilde{\pi}(d)}\,, \tag{3.27}$$

where y_d is the binary relevance label of document d and $\tilde{\pi}(d)$ is it's position for ranking $\tilde{\pi}$. The mean of the reciprocal ranks over all queries q is the MRR [LÖ09].

To define the Mean Average Precision, the Precision at position k (P@k) needs to be defined first. Given the binary relevance labels y_d, the P@k determines the proportion of relevant items, that were ranked among the first k documents by the engine. Mathematically, it is defined as

$$\text{P@}k = \frac{1}{k} \sum_{d \in D_q} \mathbb{1}_{[\tilde{\pi}(d) \leq k]} y_d \tag{3.28}$$

where $\mathbb{1}$ is the indicator function, specifying whether document d was ranked among the top k positions. Since the P@k ignores the total number of relevant items, the Average Precision (AP) is used. It calculates for each relevant document associated with query q the precision up to the position the respective document was ranked on and divides it by the number of relevant items. Formally:

$$\text{AP} = \frac{\sum_{k=1}^{|D_q|} \text{P@}k \cdot \text{rel}(k)}{\sum_{d \in D_q} y_d}, \tag{3.29}$$

where $\text{rel}(k)$ is an indicator function, evaluating to one if the item ranked at position k is relevant. The mean value of AP over all queries is the MAP [Liu09].

The second family of ranking metrics uses non-binary relevance labels, and therefore must be able to measure the relative utility of a product compared to other products in the list. The most popular utility-based ranking metric is the Normalized Discounted Cumulative Gain (NDCG). The NDCG is the Normalized DCG, which for a given ranking $\tilde{\pi}$ is defined as:

$$\text{DCG@}k = \sum_{d \in D_q : \tilde{\pi}(d) \leq k} \frac{2^{y_d} - 1}{\log_2(1 + \tilde{\pi}(d))} \tag{3.30}$$

In order to obtain the NDCG for a ranking $\tilde{\pi}$, the DCG for that ranking is divided by the DCG of an optimal ranking π^*.

Current state-of-the-art models utilize the LambdaLoss framework, whose core idea is to dynamically adjust a pairwise loss during training based on the absolute difference between the NDCG values when two documents i and j are swapped. Especially LambdaMART, which combines this loss function with gradient boosted decision trees, outperforms other models on traditional benchmarks like the Yahoo! LETOR dataset [CC11]. These benchmarks typically use ordinal relevance labels ranging from 1 (not relevant) to 5 (very relevant). However, in this work, binary relevance labels are used, making the NDCG inappropriate. There is not necessarily a single optimal ranking in the case of binary relevance labels. Let there be $n^+ > 1$ clicked documents in the SERP, then any ranking π that places all clicked documents on any of the first n^+ positions is optimal, leading to $n^+!$ "best" rankings. This makes the NDCG ill-defined for ranking with binary relevance labels. Another shortcoming of the Lambda Framework is the limitation to pairwise-loss functions. Since we utilize the whole SERP in the scoring function through the MHSA, it also makes sense to use a loss function that explicitly considers the entire SERP.

Therefore, this work uses the ListNet loss, a listwise loss function proposed by Cao et al. (2007). The ListNet loss is similar to the softmax cross-entropy loss with the difference that also the relevance labels for a list of documents are normalized

by a softmax to make it applicable to multi-level labels. Since this work uses binary labels, these are simply divided by their sum over the products in the SERP. In order to also account for cases where the label is zero for all products in the list, let the normalized labels be defined as:

$$\Pr_y(i|\mathbf{y}) = \begin{cases} 0, & \text{if } \sum_{j=1}^{m} y_j = 0 \\ \frac{y_i}{\sum_{j=1}^{m} y_j}, & \text{otherwise} \end{cases} \tag{3.31}$$

where m again is the length of the SERP. And, similar to equation 3.17, let the probability of an item i to be ranked on the top of the SERP by ranking function f be:

$$\Pr_s(i|X,\theta) = \frac{\exp(f(\mathbf{x}_i\,;\,\theta))}{\sum_{j=1}^{m}\exp(f(\mathbf{x}_j\,;\,\theta))} \tag{3.32}$$

resulting in the cross-entropy loss of the form:

$$\mathcal{L}(X,\mathbf{y}\,;\,\theta) = \frac{1}{m}\sum_{i=1}^{m} -\Pr_y(i|\mathbf{y})\log(\Pr_s(i|X)) \tag{3.33}$$

In a recent study, Bruch et al. (2019) proved that the ListNet loss is a lower bound on the MRR when given binary relevance judgments, and thus optimizing (3.33) is implicitly also optimizing an important binary ranking metric.

Two things to note here. First, this formulation of the ranking loss only involves a single label y. Since the goal of this work is to develop a ranking that satisfies different user intents, orders and clicks are used as relevance signals. Therefore, chapter 3.3.3 presents a method to incorporate multiple labels in the proposed ranking framework. Second, the relevance labels y^o and y^c provided by the click-logs are biased by the ranking order, since higher ranked products have a higher probability of being examined and consequently of being clicked and purchased. So far, the positional information $\mathbf{p}$ have not been considered in the LTR framework. Therefore, chapter 3.3.2 introduces the unbiased LTR framework developed in this work.

3.3.2 Eliminating Position Bias

The most common method to eliminate the position bias from implicit relevance feedback gathered through click-logs is the inverse propensity framework proposed by Joachims et al. (2017). This method uses the inverse of a document's examination probability as sample weight in the loss function. Thus, documents on lower ranks have a larger influence on the parameter updates during training, which leads to an unbiased estimator as Joachims et al. (2017) prove. However, most authors using this approach assume the examination probability to be known [Ai19; Bru19a]. Others estimate it via search result randomization ([JSS17]), which harms the customer experience and is consequently not preferred.

As mentioned in chapter 2.3, the position bias can also be estimated using the EM algorithm. The EM algorithm iteratively updates s_{qd} and β_p with an Expectation and a Maximization step. According to Chuklin et al. (2015), the Expectation step at iteration $t+1$ is defined by the following set of equations, representing the probabilities for each possible state in the position bias model of equation (2.4):[2]

$$\begin{aligned}
\Pr(E=1,R=1|C=1,q,d,p) &= 1 \\
\Pr(E=1,R=0|C=0,q,d,p) &= \frac{\beta_p^{(t)}(1-s_{qd}^{(t)})}{1-\beta_p^{(t)}s_{qd}^{(t)}} \\
\Pr(E=0,R=1|C=0,q,d,p) &= \frac{(1-\beta_p^{(t)})s_{qd}^{(t)}}{1-\beta_p^{(t)}s_{qd}^{(t)}} \\
\Pr(E=0,R=0|C=0,q,d,p) &= \frac{(1-\beta_p^{(t)})(1-s_{qd}^{(t)})}{1-\beta_p^{(t)}s_{qd}^{(t)}}\,.
\end{aligned} \tag{3.34}$$

Following the law of total probability, the examination probability can be written as:

$$\Pr(E=1|c,q,d,p) = \Pr(E=1,R=0|c,q,d,p) + \Pr(E=1,R=1|c,q,d,p)\,, \tag{3.35}$$

where $\Pr(E=1,R=0|\,\cdot\,)$ is non-zero only if $c=0$ and similarly, $\Pr(E=1,R=1|\,\cdot\,)$ is non-zero only if $c=1$.

It follows that the marginal probability of examination is defined as:

$$\Pr(E=1|c,q,d,p) = \begin{cases} 1, & \text{if } c=1 \\ \frac{\beta_p^{(t)}(1-s_{qd}^{(t)})}{1-\beta_p^{(t)}s_{qd}^{(t)}}, & \text{if } c=0 \end{cases} \tag{3.36}$$

Similarly, the marginal probability of relevance can be defined as:

$$\Pr(R=1|c,q,d,p) = \begin{cases} 1, & \text{if } c=1 \\ \frac{(1-\beta_p^{(t)})s_{qd}^{(t)}}{1-\beta_p^{(t)}s_{qd}^{(t)}}, & \text{if } c=0 \end{cases} \tag{3.37}$$

The Maximization step of the EM-algorithm then uses these quantities from the Expectation step to update the hidden variables by taking the observations from the click-logs $\mathcal{D} = \{(c,q,d,p)_i\}_{i=1}^n$ [Wan18]:

$$\beta_p = \frac{\sum_{(c,q,d,p')\in\mathcal{D}} \mathbb{1}_{[p'=p]} \cdot \Pr(E=1|c,q,d,p)}{\sum_{(c,q,d,p')\in\mathcal{D}} \mathbb{1}_{[p'=p]}} \tag{3.38}$$

$$s_{qd} = \frac{\sum_{(c,q',d',p)\in\mathcal{D}} \mathbb{1}_{[q'=q,\,d'=d]} \cdot \Pr(R=1|c,q,d,p)}{\sum_{(c,q',d',p)\in\mathcal{D}} \mathbb{1}_{[q'=q,\,d'=d]}} \tag{3.39}$$

[2] Note that $\Pr(E=1,R=0|C=1)$, $\Pr(E=0,R=1|C=1)$ and $\Pr(E=0,R=0|C=1)$ are zero, since a click always requires the document to be both, relevant and examined according to the click model of equation (2.4). Conversely, $\Pr(E=1,R=1|C=0)$ is zero as well.

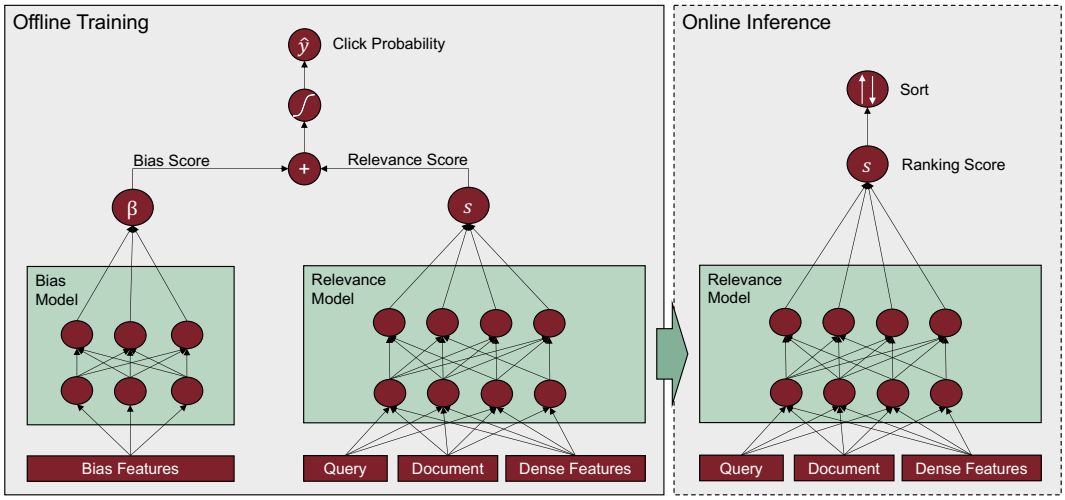


Fig. 3.6.: Architecture of the Joint Estimator (JoE)

The problem with this approach is that it requires the same (q, d)-pairs to appear multiple times in different positions, otherwise the solution is ill-defined [Wan18; CZ09]. Moreover, separating the bias estimation from the LTR task leads to inferior results compared to end-to-end approaches that learn a bias and a relevance model jointly, as shown by Ai et al. (2018). Therefore Wang et al. (2018) propose a regression-based Expectation-Maximization algorithm that jointly learns propensity and ranking scores based on click data. Instead of working with (q, d) identifiers, this algorithm expects a feature vector $\mathbf{x}_{qd}$ representing (q, d) based on which it models the relevance as a function $f(\mathbf{x}_{qd}) = s_{qd}$. The major drawback of this approach is that the parameter updates during the Maximization step are computed per query-document pair and consequently cannot consider the whole list of products [Hu19]. As a result, the relevance function f can only be trained with pointwise losses, making it inapplicable to the LTR framework using the ListNet loss proposed in this work.

Inspired by the regression EM algorithm, this work proposes another method to jointly learn a ranking model and a position bias estimator, which is not restricted to the pointwise setting. Instead of using a regression function only for the relevance estimation, this method also models the position bias β as a function $f_\beta(\mathbf{p})$ of positional information $\mathbf{p}$. As for the relevance model, the position bias is modeled by a neural network. In this work, the hidden layers are defined as a stack of fully connected layers and ReLU activation functions. The output layer consists of a single node per product and no activation function. Moreover, dropout and batch normalization are used after each hidden layer. In the end, the output of the relevance model and the position bias model are combined as follows:

$$s_{qd} = f(\mathbf{x}_{qd}; \theta) \tag{3.40}$$

$$\beta_{qd} = f_\beta(\mathbf{p}_{qd}; \gamma) \tag{3.41}$$

$$\hat{y}_{qd} = \sigma(s_{qd} + \beta_{qd})\,, \tag{3.42}$$

where the final prediction $\hat{y}$ directly models the click probability $\Pr(C = 1)$ of equation (2.4), while the outputs of the subnetworks $f_\beta(\mathbf{p}_{qd})$ and $f(\mathbf{x}_{qd})$ approximate $\Pr(E = 1)$ and $\Pr(R = 1)$ respectively. With the use of neural networks for both the relevance and the position bias estimator, the respective parameters θ and γ can be learned through backpropagation by evaluating the loss function based on the combined score $\hat{y}$. Note that the scores of the bias and the relevance model are added, while the click model of equation (2.4) assumes a multiplicative relationship. This is because the scores s_{qd} and β_{qd} are on a logarithmic scale and are mapped to a probability space by the softmax activation in the ListNet loss only after the addition. Since an addition in logarithmic space corresponds to the product of probabilities, i.e. $\mathrm{pr}_1 \cdot \mathrm{pr}_2 \equiv \log(\mathrm{pr}_1) + \log(\mathrm{pr}_2)$, the output of the bias model is added to the relevance score.

While this approach is comparable to incorporating the bias-related features in the relevance model, it does not have the limitation that the positional information is required during inference, where it is no longer available. When scoring new documents, only the relevance estimation is important and consequently, the position bias model can be ignored. Figure 3.6 depicts the architecture of this Joint Estimator (JoE) and shows the difference in model training and inference.

Moreover, by utilizing a functional form for β, the position bias estimation is no longer limited to the position p. This is an important advantage of the proposed method, as online stores usually present their products with a variety of layouts, the choice of which depends on the user's device type and the taxonomy of the presented products (see figure 3.7). This is in contrast to the list-based web search scenario, where results are exclusively presented from top to bottom. Guo et al. (2020) investigate the position bias in a grid-based e-commerce environment and found that the device and layout types have a significant impact on users' shopping behavior. Therefore, this work also incorporates layout and end device information in the feature space P of the position bias estimator. In addition, one could also present features like the specificity of a query to the bias model, to account for the different user intents like buying and browsing. However, extensive research of relevant bias features will be left for future research.

3.3.3 Multi-Task Learning

As described in sec 2.4, training a neural ranking model for the product ranking task only on clicks is problematic. Therefore, a method that is capable of handling the multiple relevance signals prevalent in e-commerce must be incorporated in the proposed LTR framework. In the LTR literature, only two methods that deal with multiple objectives have been identified. First, the 3-part hinge loss proposed by Nigam et al. (2019), which tries to score clicked and purchased products higher than clicked but not purchased products. One issue with this approach is that its quality largely depends on the threshold values for the distinct classes, which have to be tuned manually. Moreover, the hinge loss poses a pointwise loss function, which does

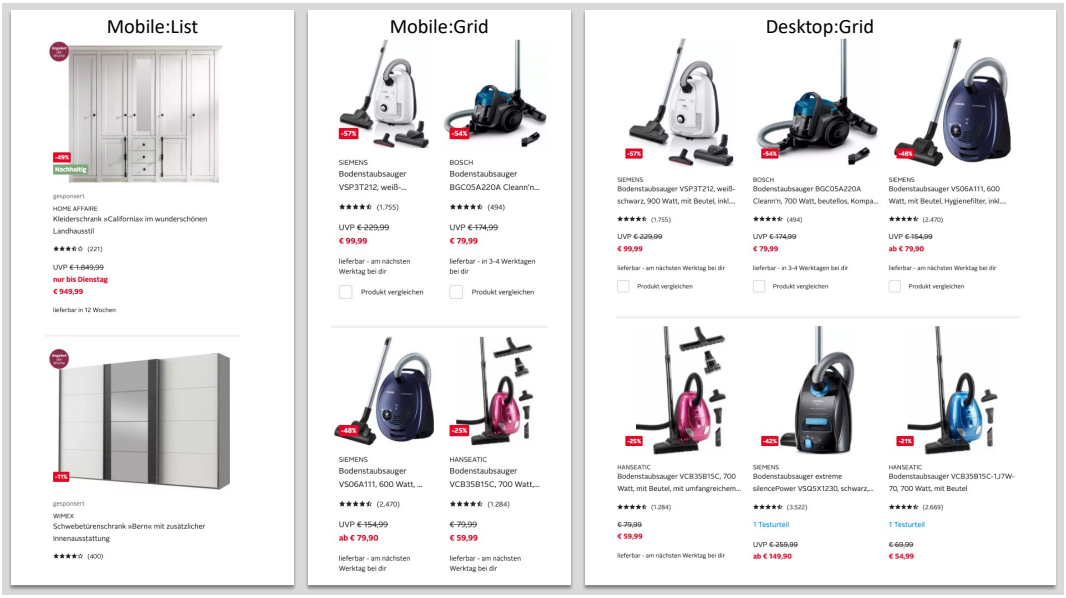


Fig. 3.7.: Illustration of the different layouts prevalent in online shops

not correlate with ranking metrics. And using a single model to optimize both tasks might also be problematic when there are conflicting relationships.

Therefore, Wu et al. (2018) model the order prediction task independently from the click prediction task as a pointwise classification problem. However, we argue that modeling the order prediction task as a pointwise classification problem is too simplistic. Even if the user only sees the item of interest after clicking on it (and not the SERP anymore), the other items in the list still compete with that item for the final order. A user might click on multiple items on the SERP and, in the end, has to decide on one of those items. Moreover, the number of observed purchases might be too low to learn complex relationships.

To overcome the limitations of these methods, this work will utilize a multi-task learning framework, first proposed by Caruana (1997). The traditional multi-task learning model utilizes a shared bottom approach, where a shared layer is used as input for the task specific networks. The benefit of this framework is that the full set of training data is used to train the bottom hidden layers and only the top layers of the network are task-specific. This, of course, requires some relatedness between the tasks, however since the click is a precondition for an order, the click and order prediction tasks are sufficiently related and information extracted for one task can be useful for the other.

Having said that, one issue with the shared bottom approach is that all sub-networks have to use the same learned input representation, meaning that errors are backpropagated through the shared network for each task alike. This can harm the quality of the shared hidden layers, when the correlation between the tasks is low or conflicting relationships with the feature space exist [Ma18]. Such conflicting relationships exist between the click and order prediction tasks, as has been shown by the price example in chapter 2.4. To overcome this limitation, Ma et al. (2018) developed the Multi-Gate Mixture-of-Experts (MMoE) framework. The core of the

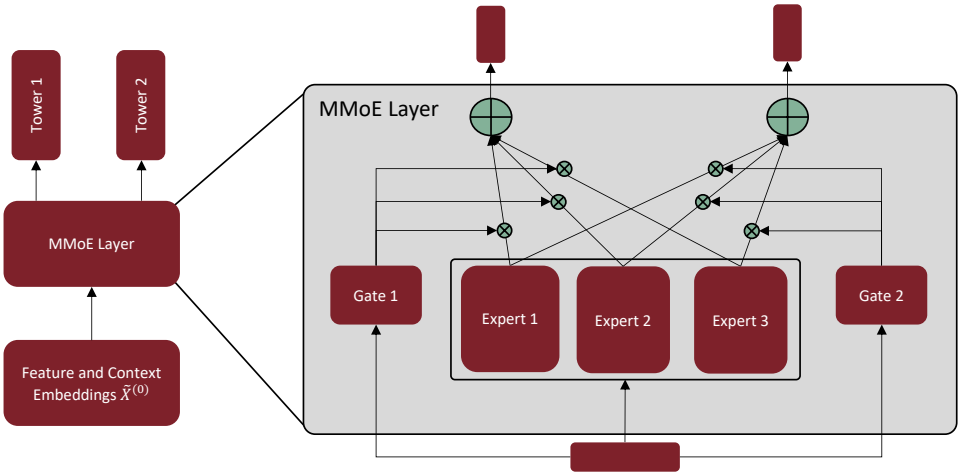


Fig. 3.8.: A Multi-Gate Mixture-of-Experts (MMoe) Layer as described by Ma et al. (2018) implemented after the Latent Cross of Feature and Context Embeddings

MMoE layer are multiple expert networks $f_i|_{i=1}^{N^E}$, each of which focuses on different parts of the input. Similar to Ma et al. (2018), we use a single hidden layer of fully connected units and ReLU activation for each expert network. The same input of the expert networks is passed through K task-specific gating networks, similar to the one implemented in equation (3.7). Similar to the GRU-DSSM, the gating network g^κ of task κ determines how much each expert influences the output for that task. Formally, the MMoE layer can be described as follows:

$$\begin{aligned} S^\kappa &= f^\kappa(h^\kappa(\tilde{X}^{(0)}))\,, \qquad \forall \kappa = 1, ..., K \\ \text{where } h^\kappa(X) &= \sum_{i=1}^{N^E} g_i^\kappa(X) f_i(X) \qquad \forall \kappa = 1, ..., K \end{aligned} \tag{3.43}$$

and $\tilde{X}^{(0)}$ is the result from the latent cross operation of the ranking function defined in equation (3.25). Lastly, $f^\kappa(\cdot)$ are the task-specific tower networks. This work employs several hidden layers of fully connected units, ReLU activation functions, dropout and batch normalization to build each tower network. The last layer comprises a single output node per product in the SERP and no activation function is used to produce the task-specific relevance scores s^κ. Then, the position bias scores β are added to the task-specific relevance scores to form the logits $\hat{y}^\kappa$.

It follows that the ranking framework introduced in chapter 3.3.1 has to be reformulated to incorporate the described multi-task learning approach as well as the bias scorer f_β. Therefore, let the probability of an item being scored on top of the SERP under task κ be defined as:

$$\Pr_s^\kappa(i|X, P, \theta, \gamma) = \frac{\exp(f^\kappa(\mathbf{x}_i\,;\,\theta) + f_\beta(\mathbf{p}_i;\gamma))}{\sum_{j=1}^{m} \exp(f^\kappa(\mathbf{x}_j\,;\,\theta) + f_\beta(\mathbf{p}_j;\gamma))} \tag{3.44}$$

Then, the losses for the click and order prediction task can be determined as follows:

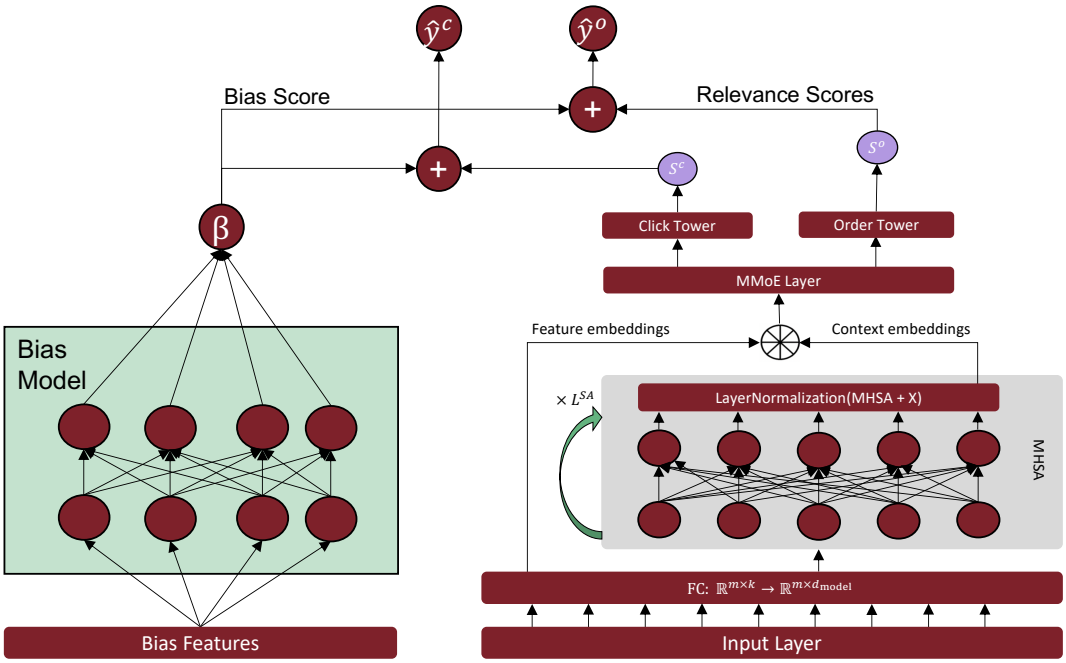


Fig. 3.9.: The complete unbiased neural ranking architecture

$$\mathcal{L}^c(X, \mathbf{y}^c\,; \theta, \gamma) = \frac{1}{m}\sum_{i=1}^{m} -\mathrm{Pr}_y(i|\mathbf{y}^c)\log(\mathrm{Pr}_s^c(i|X, P, \theta, \gamma))$$
$$\mathcal{L}^o(X, \mathbf{y}^o\,; \theta, \gamma) = \frac{1}{m}\sum_{i=1}^{m} -\mathrm{Pr}_y(i|\mathbf{y}^o)\log(\mathrm{Pr}_s^o(i|X, P, \theta, \gamma)) \tag{3.45}$$

The ListNet loss evaluates to zero if the labels for all products in the SERP are zero. Since orders occur very rarely compared to clicks, the loss for the order prediction task is zero most of the time. Simply adding the losses for both tasks would lead to the model focusing almost exclusively on the click prediction task, since the average loss is much higher on this task. To overcome this issue, this work proposes a strategy for combining two different losses with different magnitudes in multi-task learning. Given two losses $\mathcal{L}_1$ and $\mathcal{L}_2$ for the click and order prediction task, respectively, the combined loss will be defined as

$$\tilde{\mathcal{L}}(\mathcal{L}_1, \mathcal{L}_2) = \lambda\mathcal{L}_1 + (1-\lambda)\mathcal{L}_2 + |\lambda\mathcal{L}_1 - (1-\lambda)\mathcal{L}_2|\,, \tag{3.46}$$

where λ is a learnable parameter that is optimized during training. The term $|\lambda\mathcal{L}_1 - (1-\lambda)\mathcal{L}_2|$ is the absolute difference between the two weighted losses and acts as a regularization term, which ensures that the learning algorithm does not focus on one task over the other during training. To validate this, the following theorem is formulated:

Theorem 1. *Given two losses $\mathcal{L}_1$ and $\mathcal{L}_2$, a learning algorithm minimizing a loss function of the form $\tilde{\mathcal{L}} = \lambda\mathcal{L}_1 + (1-\lambda)\mathcal{L}_2 + |\lambda\mathcal{L}_1 - (1-\lambda)\mathcal{L}_2|$ also minimizes the absolute difference between the weighted losses $\lambda\mathcal{L}_1$ and $(1-\lambda)\mathcal{L}_2$.*

And the proof is given in Appendix A. From the concept of empirical risk minimization it follows that the set of parameters $\Theta = \{\theta, \gamma, \lambda\}$ can be learned by solving the following optimization problem:

$$\underset{\Theta}{\operatorname{argmin}}\, \hat{\mathcal{R}}(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{(X,P,\mathbf{y}^c,\mathbf{y}^o)\in\mathcal{D}} \tilde{\mathcal{L}}(\mathcal{L}^c, \mathcal{L}^o; X, P, \mathbf{y}^c, \mathbf{y}^o, \Theta) \tag{3.47}$$

with the combined loss $\tilde{\mathcal{L}}$ defined in equation (3.46). In the prediction stage, the final ranking score $\hat{y}$ is determined by a weighted average of the click and order ranking scores:

$$\hat{y} = w^c \hat{y}^c + w^o \hat{y}^o \tag{3.48}$$

where the optimal weights w^c and w^o for the click and order scores are determined via grid-search. This is described in detail in chapter 4.2.2. Finally, the complete network architecture is summarized in figure 3.9.

4 Empirical Results

> *If the users do not trust a model or a prediction, they will not use it*
>
> — **Marco Tulio Ribeiro**
> (Researcher at Microsoft)

In this section, we will evaluate the quality of the proposed methods. Therefore, we first conduct an extensive offline evaluation for all three model components described in chapter 3. For each component, the approach that achieves the most promising results offline is used in the final neural ranking model. We then compare the resulting rankings against the status quo ranker as part of an online A/B test on otto.de. The scale of the Otto online shop makes it an ideal test environment for the proposed ranking system. With a revenue of 4.5 billion euros, Otto is the second largest online shop in Germany [Ott22]. Also, over 11 million active customers and an average of 50 million daily visitors on otto.de allow us to collect enough customer interactions for training and testing the proposed methods.

4.1 Implementation Details

To train the encoder as well as the ranking function, we use click-logs from otto.de over a period of three weeks, where the last two days are used as a validation set. We only consider search-related events, i.e. customer interactions with products displayed on the SERP after a query via the search engine of otto.de has been issued. This excludes external jumps on SERPs, e.g. via Google and cases where a customer entered a SERP by navigating through the shop. Moreover, bot sessions and query sessions in which no clicks occurred are filtered out.

Table 4.1 summarizes the statistics of the resulting dataset. The column *Searchterms* refers to the number of distinct textual queries users have issued, while the number of *Searches* is the total number n of received queries for which a ranking has been returned. The column *Total Samples* refers to the total number of query-document pairs for which implicit user feedback has been collected. The *Clicks* and *Orders* columns show how many of these were clicks or orders, respectively.

Tab. 4.1.: Statistics of the Dataset

Type	Days	Searchterms	Searches	Total Samples	Clicks	Orders
train	19	1.830.594	9.640.653	491.086.405	26.468.803	601.359
test	2	335.450	1.053.492	54.465.341	2.934.554	62.362

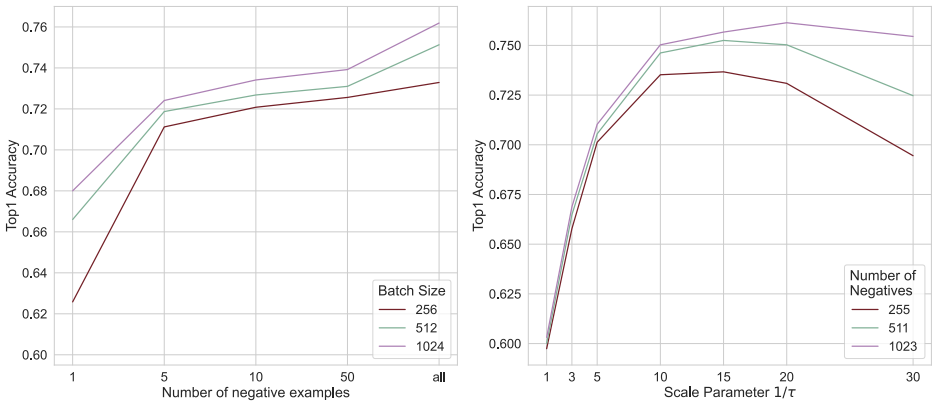


Fig. 4.1.: Influence of the number of negative examples (left) and of the temperature hyperparameter τ (right) on the accuracy

To handle the immense amount of data, the dataset generation and preprocessing are conducted using Google's BigQuery and Apache Spark. All models are implemented using the deep learning framework Tensorflow. The training and validation data are transformed into protocol buffers, which allow fast read operations of the data. This way, we achieved a GPU utilization of around 90% during training and, as a result, were able to train the ranking model in less than two days of training time.

4.2 Offline Evaluation

4.2.1 Encoder Evaluation

The encoder is meant to provide the scoring function with the ability to determine the semantic similarity between the query and the product description beyond exact term matches. One critical aspect of the encoder training is the mining of (hard) negative examples, which ultimately ensures that the encoder learns to discriminate matching from non-matching products for a given query. Therefore, as a first step, this work compares the different approaches for training an encoder network described in chapter 3.1.3. First, the influence of the number of sampled negatives per positive example as well as the size of the batch is evaluated. For the case of one negative per positive example, the triplet loss of equation 3.9 is utilized and the product whose embedding is the closest to that of the query is selected as the negative example. In the case of multiple negatives, the softmax loss of equation 3.12 is used and again the items that are the closest to the query in the embedding space are chosen as negatives. Moreover, the case where all other products in the batch are used as negatives is evaluated.

The plot on the left of figure 4.1 shows the performance of the DSSM model for the different training strategies by means of the (Top1) accuracy on the validation set. The accuracy measures the fraction of predictions in which the matching product was correctly identified as the positive example, i.e. received the highest score among all

Tab. 4.2.: Comparison of the different encoder architectures

Model	Contrastive Learning Task		PBK Prediction
	Top1 Accuracy	**Top5 Accuracy**	**Top1 Accuracy**
DSSM	0.7516	0.9749	0.9911
GRU-DSSM	0.7711 (+2.6%)	0.9807 (+0.6%)	**0.9975 (+0.6%)**
USE	**0.7761 (+3.3%)**	**0.9817 (+0.7%)**	0.9973 (+0.6%)

products. More generally, the Top$K$ accuracy measures the fraction of predictions in which the matching product is among the top K products with the largest predictions.

All methods are evaluated on the same validation set, which comprises of batches of size 256, i.e. 255 potential negative examples for each product. One can see that the best results are achieved using the softmax loss with all other items of the batch used as negative examples. Moreover, similar to Chen et al. (2020), we found larger batch sizes to be helpful in learning discriminative embeddings, especially when all other items of the batch are used as negatives. However, larger batch sizes also lead to higher computational requirements, which is why we were not able to test batches with a size larger than 1024. The triplet loss performed the worst among all training strategies, with the performance gap being the largest for smaller batches.

Since the performance of the softmax loss largely depends on the temperature parameter τ, this hyperparameter is tuned on the validation set. The results for three example values of N^N are shown in the right plot of figure 4.1. One can see that the more negative examples are used the larger the scaling factor $1/\tau$ should be chosen, which is in conjunction with the behavior of the softmax upper bound seen in figure 3.2.

The performance of the different training strategies has been evaluated on the DSSM model. In the following, the quality of the DSSM encoder is compared with the proposed GRU-DSSM architecture as well as the transformer-based Universal-Sentence-Encoder (USE) described in chapter 3.1. All models use the same wordpiece tokenizer described in algorithm 1 with a vocabulary size of 30,000 tokens. The dimensionality of the token embeddings is set to 512 for all architectures. Moreover, the dimensionality of the hidden states of the GRU layer as well as the d_{model} parameter of the USE are set to 512. The DSSM and GRU-DSSM architectures use an MLP on top of the pooled token embeddings with two hidden layers of 300 fully-connected units each. The USE consists of 2 layers of MHSA layers with 4 attention heads. This results in around 250,000 trainable parameters for the DSSM, 760,000 for the GRU-DSSM and 4.2 million for the USE (excluding the embedding layer).

Table 4.2 shows the performance of the different architectures based on the Top1 and the Top5 accuracy on the contrastive learning task as well as the Top1 accuracy on the PBK prediction task, each of which is evaluated on the validation set. Moreover, the relative change of the metrics compared to the DSSM is given in parenthesis. While the more complex models achieve a fairly substantial performance improvement

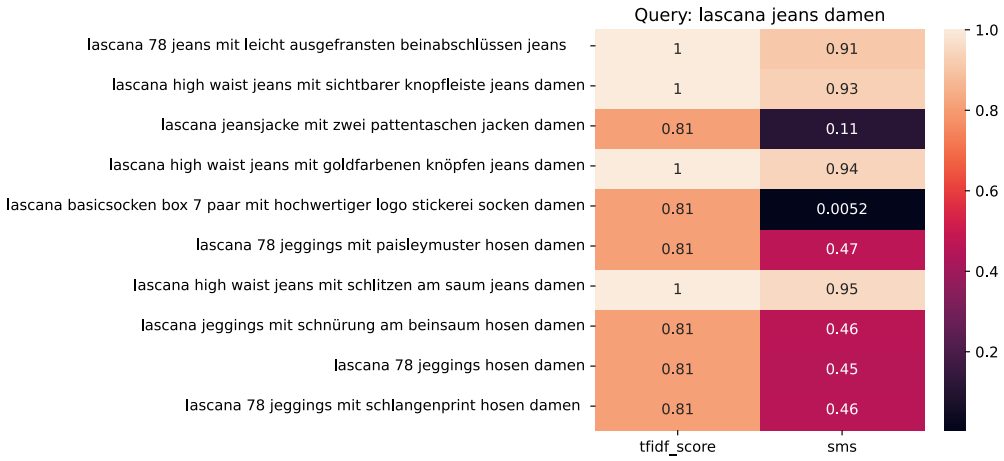


Fig. 4.2.: Comparison of TF-IDF score and the semantic matching score (sms)

over the DSSM architecture, the differences between the GRU-DSSM and the USE are not too large. The USE is performing better on the contrastive learning task and the GRU-DSSM achieves a slightly higher accuracy on the PBK prediction task. What stands out is the very high accuracy over all models on the PBK prediction task. Still, the 0.6% improvement of the GRU-DSSM and the USE compared to the DSSM is remarkable since the accuracies are already above 99%. Especially on the PBK prediction task, the ability of the more complex encoders to learn context-dependent embeddings and assign different weights to those is important since the PBK can often be inferred from a single token in the product title. However, the DSSM still achieves very good results on both tasks, indicating a good fitness of all these models for the encoding task. Due to the negligible performance improvements compared to the immense increase in computational costs of the transformer-based model, the GRU-DSSM model will be used as the encoder in the remainder of this work.

The main purpose of the encoder is to determine the semantic relevance of a given product to a user query. Historically, this has been measured using hand-crafted features, of which TF-IDF and BM25 are the most successful and popular ones. However, these features not only measure exact matches, thus ignoring synonyms, misspellings or word combinations, which are very common in the German language (for example “Kleid” will not match “Sommerkleid”). To evaluate whether the GRU-DSSM is successful in overcoming these shortcomings, the cosine similarities between query and document embeddings have been compared to the sum of TF-IDF scores over query tokens for the same documents. Figure 4.2 shows the result for an example query and the corresponding retrieval set.

What stands out is that the variance of the cosine similarities is much higher than that of the TF-IDF scores. The TF-IDF matches the brand name “lascana” as well as the target group (“damen”) in each product title and consequently takes on rather high values for each of the products. The TF-IDF scores for the relevant products at positions 1, 2, 4 and 7 are the largest, because they match the additional query term “jeans” as well. Hence, the TF-IDF scores provide good guidance toward the most

Tab. 4.3.: Cosine similarities for examples of query-document pairs

Query	Product	Kind	$\mathbf{v}_q^\top \mathbf{v}_d$
iphone 13	iphone 13 pro max 512 gb	+	0.9245
	handyhülle klar silikon kompatibel mit iphone 13	-	0.3625
küche ohne e geräte	küchenzeile ohne e geräte breite 360 cm	+	0.9891
	küche mit hanseatic e geräten wahlweise mit kühlschrank	-	0.5251
spiel für die switch	mario kart 8 deluxe nintendo switch	+	0.8094
	nintendo switch spielekonsole	-	0.5035

relevant documents. However, the TF-IDF fails at distinguishing between the lesser relevant products, where its scores are all the same. However, a good ranking system should rank "jeggings" (pants) higher for a query "jeans" than jackets (position 3) or socks (position 5). The cosine similarity between the encoder embeddings for query and documents achieves just that. While also being the highest for the relevant products, the cosine similarity still takes on rather high values for the pants (about 0.5) and low values for the completely irrelevant products (socks and jackets in this case).

Lastly, table 4.3 shows the cosine similarity between the GRU-DSSM embeddings of different queries and one positive and a hard to classify negative example each. In each case, the encoder manages to place the positive example closer to the query in the embedding space, yet the negative examples are not too far away from the query either.

4.2.2 Ranker Evaluation

Next, the scoring function is evaluated. Besides the cosine similarity between query and product description embeddings, which will be referred to as semantic similarity score (sms), and the matching score of a product's pbk with the predicted user intended pbk, several hand-crafted features listed in table B.1 are used as input for the scoring function. To learn the parameters of the scoring function, Adam is used as optimization algorithm with a variable learning rate of the form:

$$lr = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}), \qquad (4.1)$$

which is a common choice for training transformer based models [Vas17]. $warmup\_steps$ is a hyperparameter which we choose to be 8,000. $step\_num$ corresponds to the number of batches that have been seen during training. This work uses a batch size of 128, resulting in more than 70,000 training steps per epoch. We train the model for 5 epochs, which corresponds to roughly 350,000 parameter updates. Moreover, the number of products per query is fixed to 72, which corresponds to the number of products shown on the first page of the SERP. Longer SERPs are truncated at the end and shorter SERPs are padded, where masking is used to ignore padded values.

(a) Feature importance (clicks)

(b) SHAP values for the click prediction task

(c) Feature importance (orders)

(d) SHAP values for the order prediction task

Fig. 4.3.: SHAP values for the click (top) and order prediction task (bottom)

In a first step, the influence of the features on the click and order predictions will be evaluated using SHAP, short for SHapley Additive exPlanation [LL17]. SHAP values quantify the contribution of features to the model prediction by inspecting how the prediction would change if the feature would take on some default value. Figure 4.3a shows the mean average SHAP value per feature on the click prediction task, where large values imply great importance of a feature in terms of its influence on the final prediction. One can see that the number of past clicks on a product is by far the most influential feature for the click prediction task, followed by the cosine similarity of query-product embeddings. Interestingly, the price, shipping costs or delivery time have nearly no impact on the click behavior.

In figure 4.3b, one can see the SHAP values of the different features per query-document pair, which give an indication of the relationship between the value of a feature and the model prediction. Each dot represents an instance of the validation set, and its color specifies the instance's value for the respective feature. When high values of a feature correspond to negative SHAP values for many observations, this implies a negative correlation between the respective feature and the model output and vice versa. Using the SHAP values, assumptions regarding the effects of certain features on the model prediction can be validated. Generally, the SHAP values reflect the effects that are to be expected for most features. For example, a lot of clicks on a product in the past imply a high click probability in the feature. The same holds true for high semantic similarity or the number of reviews a product received. On the other hand, old products (large values for freshness) have a lower probability of being clicked. Interestingly, while the price does not have a large influence on the click probability for most products, a high price seems to increase the ranking score on the click prediction task, which is in line with the findings from chapter 2.4.

Regarding the order prediction task, the influence on the model prediction changes for many features. As expected, the most important feature for the order prediction is a product's previous sales record (see figure 4.3c). Surprisingly, however, the number of previous clicks on a product has only a mediocre effect on order prediction and the effect is reversed compared to the click prediction task (see figure 4.3d). This highlights the need to use the MMoE layer instead of the shared-bottom approach, which cannot capture such contradictory relationships. We also observe an increase in the importance of the price, which is negatively correlated with orders, as expected. The same is true for delivery time and shipping costs, both of which have virtually no effect on the click. Another notable observation is the influence of query occurrences, which has a very strong impact on the order prediction. The more often a search term occurs, the less likely a user is to make a purchase following the query. We investigated this and found that the most frequently occurring search terms are very

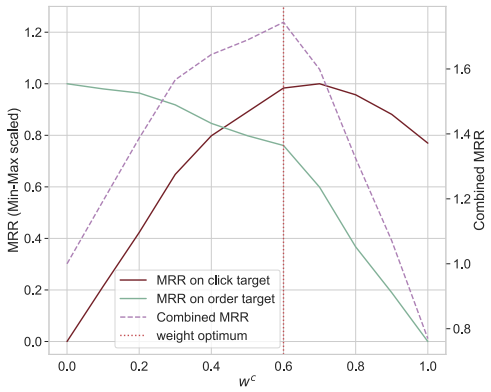


Fig. 4.4.: Optimal weights for the click and order prediction tasks

Layer	Hyperparameter	Values
MHSA	# MHSA layers L^{SA}	{2, 3, **4**}
	d_{model}	{256, **512**}
	# attention heads h	{2, **4**, 8}
MMoE	# experts	{2, 3, **4**}
	# tasks	{1,**2**}
	# expert hidden units	{256, **512**}
Tower	# hidden layers	{2,3,**4**}
	# hidden layer units	{**512**}
	dropout	{0.0, **0.2**, 0.5}

Tab. 4.4.: Hyperparameter-Tuning

non-specific, such as “sofa” or “carpet”. Users who issue such a query probably do not have a clear product in mind and thus no intention to order, but rather want to get inspired.

One critical aspect of the proposed multi-label ranking function is the combination of the different task-specific relevance scores to a final ranking score. To choose the respective weights w^c and $w^o = (1 - w^c)$, the performance of the ranking model is evaluated for different weight combinations on the validation set. Then the weights are chosen such that the combination of the task-specific MRRs is maximized. However, since the MRR of clicks is generally higher than that of orders, the click prediction task would have a larger influence than the order prediction task when simply adding their respective MRRs. Therefore, a min-max scaling is performed beforehand. The result can be seen in figure 4.4.

Lastly, the proposed ranking model is compared against the current production ranker, which is described in chapter 4.3.1 as well as a simple benchmark. For the benchmark, the MHSA layers are omitted and the original input vector $\mathbf{x}$ is passed to the MMoE layer directly, so that the effect of the cross-attention mechanism can be validated in isolation. Therefore, the benchmark is also trained using the ListNet loss. The models are compared using the MRR and MAP on the validation set.

The hyperparameters of the different layers of the scoring function, i.e. the MHSA, MMoE and the task-specific towers, are listed in table 4.4. Grid search with the specified value options is performed to tune the respective hyperparameters for the benchmark and the AttnRank model. The parameter values that achieved the best offline result are written in bold, where no difference in the optimal hyperparameters between the benchmark and the AttnRank architecture was observed.

Note that in order to measure the influence of multiple feedback signals, we also evaluate the single-task variants where only the click is used as a relevance label. For one task, the MMoE layer simply reduces to another hidden layer of fully connected units. For the multi-task case, the method described above is used to determine the optimal weights w^c and w^o for each model. The results in terms of the MRR and MAP metrics obtained on the validation set for both the click and the order

Tab. 4.5.: Offline Results for the Scoring Functions

Model	MRR click	MRR order	MAP click	MAP order
Status Quo	0.493	0.2559	0.3264	0.2365
Benchmark on clicks	0.5099	0.2201	0.3360	0.1999
Benchmark on clicks & orders	0.5140	0.2570	0.3455	0.2393
AttnRank on clicks	0.5154	0.2216	0.3464	0.2061
AttnRank on clicks & orders	**0.5299**	**0.2649**	**0.3562**	**0.2467**

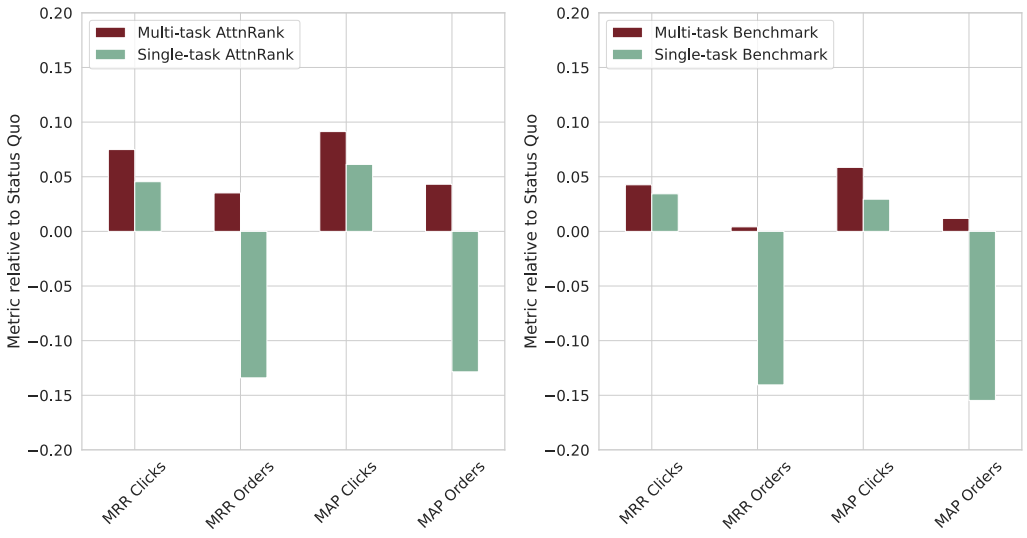


Fig. 4.5.: Performance of the AttnRank (left) and the benchmark model (right) relative to the production ranker

labels are presented in table 4.5. Interestingly, the click-related metrics improve in the multi-task setting for both models. This is noteworthy since the single-task uses only clicks as labels, yet cannot beat the multi-task approach on this very target. In fact, this is in line with the findings of Ma et al. (2018) and can be attributed to the fact, that the tasks are related and thus, both signals provide meaningful information for the other task in each case. Moreover, both the benchmark and the proposed AttnRank model achieve better results than the production ranker on all metrics when trained on orders and clicks. In this regard, the addition of the order prediction task yields a strong increase in the order-related metrics. Overall, the AttnRank model trained on both clicks and orders is the best-performing model. Figure 4.5 illustrates these results relative to the status quo ranking, indicating big improvements over the status quo. Especially due to the fact that the status quo benefits from the position bias, these findings show the effectiveness of the presented approaches. However, since we cannot infer what the users would have clicked or purchased when shown the proposed rankings, these findings have to be confirmed in an online A/B test, which is conducted in chapter 4.3.

4.2.3 Position Bias Evaluation

Lastly, the proposed method to estimate and isolate the position bias is evaluated. Since the position bias is an unobservable quantity, we use synthetic data to evaluate this part of the ranking model. This work follows the approach described by Joachims et al. (2017), who use the Yahoo! LETOR dataset to simulate biased clicks. The Yahoo! LETOR dataset is a publicly available LTR dataset with a total of 29,921 queries. Each query-document pair has an integer relevance label ranging from 1 (not relevant) to 5 (very relevant) and is described by 700 features. To simulate clicks on this dataset, Joachims et al. (2017) first use 1% of the data to train a ranking model, which mimics the production ranker and generates SERPs π for each query in the dataset. Then, clicks on documents are simulated by using the positions the respective documents have been ranked on as well as their ground-truth relevance labels available from the original dataset. More specifically, the examination probability is modeled as $\Pr(E = 1|p) = (1/p)^{\xi}$, where ξ is used to control the severity of the position bias and is set to 0.5 if not mentioned otherwise. Moreover, the relevance probability is defined as:

$$\Pr(R = 1|q, d) = \frac{2^{y_{qd}} - 1}{2^{y_{\max}} - 1}\,, \tag{4.2}$$

where y_{qd} is the ground-truth relevance label of document d for query q and $y_{\max}$ is the best possible relevance, which is 5 in this case. Then, following the click model from equation 2.4, the probability of a click is obtained by multiplying the examination and the relevance probabilities. However, as described in section 3.3.2 the position bias for the grid layouts prevalent in online shops does not only depend on the position p, but also on the layout type. One benefit of the proposed JoE framework is that it can easily incorporate additional features like the layout and

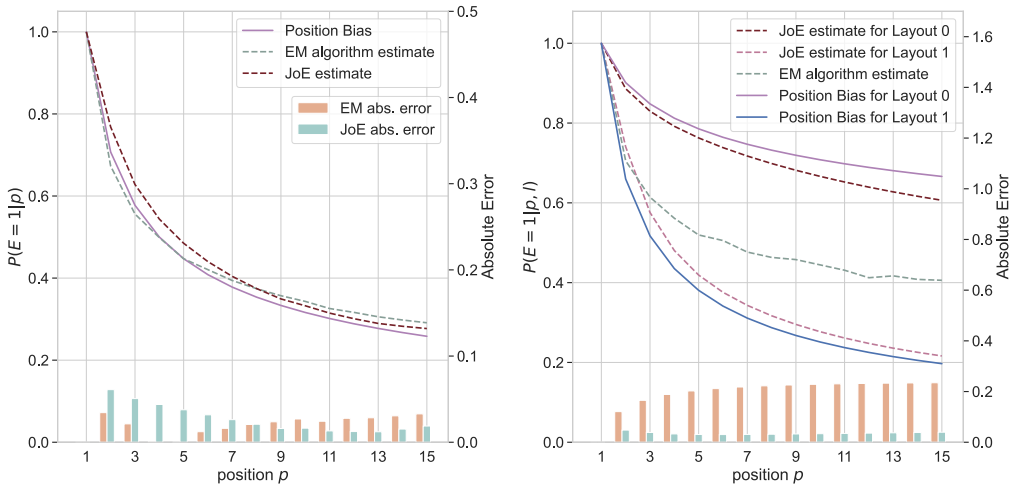


Fig. 4.6.: Comparison of the true position bias and its estimate via the EM algorithm and JoE for a single layout (left) and multiple layouts (right)

end device information. To evaluate the effectiveness of this approach, we specify a second model for the examination probability:

$$\Pr(E = 1|p, l) = \begin{cases} (1/p)^{0.15}, & \text{if } l = 0 \\ (1/p)^{0.6}, & \text{if } l = 1 \end{cases} \tag{4.3}$$

with l being a binary variable specifying the layout type. Joachims et al. (2017) generate a click dataset by repeatedly drawing search queries along with the corresponding SERPs generated by the "production ranker". Then, for each associated document, a click is simulated by conducting a Bernoulli experiment with the inferred click probabilities specifying the probability of success. This work uses $n = 40,000$ sampled queries to generate a dataset, where for each query the layout l is chosen randomly with uniform probabilities.

Now, the generated data can be used to evaluate whether the bias estimator is capable of learning the true position bias. Moreover, the performance of the proposed JoE architecture can be compared to other counterfactual LTR methods by comparing the NDCG of the predicted rankings using the ground-truth relevance labels from the original dataset. This work uses the EM algorithm described in chapter 3.3.2 as benchmark for the position bias estimation. To evaluate the effectiveness of the JoE approach on the ranking task, the inverse propensity scoring (IPS) framework with propensities determined by the EM algorithm is used as a benchmark. Moreover, the results are compared to the naive approach, which trains the scoring function on the click labels directly. In order to evaluate the effect of the counterfactual LTR approaches in isolation, each approach uses a simple Multi-layer Perceptron as scoring function and is trained using the ListNet loss.

Figure 4.6 shows the position bias estimates of the EM algorithm and the Joint Estimator for both, a single layout with the true position bias $\Pr(E = 1|p) = (1/p)^{0.5}$ and a multi-layout scenario as described in equation (4.3). For the case of a single

Tab. 4.6.: Offline Results for the (unbiased) LTR algorithms

Model	Single-Layout			Multi-Layout		
	NDCG@1	**NDCG@3**	**NDCG@10**	**NDCG@1**	**NDCG@3**	**NDCG@10**
Naive	0.6094	0.6280	0.7552	0.6265	0.6493	0.7655
IPW	**0.6704**	**0.6736**	**0.7812**	0.6592	0.6690	0.7804
JoE	0.6622	0.6725	0.7791	**0.6667**	**0.6743**	**0.7827**

layout, the EM algorithm's estimate of the position bias is slightly more accurate on the first positions than that of JoE, as can be seen from the error bars below the position bias (estimates). However, all in all, both methods manage to accurately predict the position bias with a mean absolute error below 0.03. In the multi-layout case, the EM algorithm naturally performs much worse, as it cannot absorb the layout information. At least, the EM algorithm figures out the average position bias of the two layouts. The JoE, on the other hand, can easily incorporate the layout information as a feature and learn two different position biases. For both layout types, the estimate of JoE is quite accurate, with a mean absolute error below 0.07.

The results for the ranking experiment are listed in table 4.6. It shows the NDCG values when using the relevance estimates learned by the different approaches to rank the documents in the validation dataset. These results reflect those of the position bias estimation experiment. On the single-layout dataset, the IPS estimator with the position bias estimates taken from the EM algorithm outperforms the JoE architecture. On the other hand, JoE achieves more accurate relevance predictions on the multi-layout data in terms of the NDCG. In both cases, the naive approach performs much worse than the counterfactual LTR approaches. However, it performs slightly better on the multi-layout case, which is due to the position bias being less severe on average in this data.

Lastly, the position biases learned from the click-logs of otto.de are investigated. In this dataset, we distinguish between the three device types tablet, mobile and desktop. Moreover, there are several layout options, each of which behaves differently depending on the device type, as already depicted in figure 3.7. For simplicity, this evaluation focuses only on the three-column grid layout on desktop computers as well as the classical list layout, which shows all items in a single column, independent of the device. The left plot of figure **??** shows the bias scores for the grid layout. The heatmap represents the layout structure and indicates a monotonically decreasing examination probability from top to bottom. The largest decrease can be observed from the first to the second position, highlighting the importance of ranking the most relevant item at the top of the SERP. Moreover, the plot indicates a sharp decay of the examination probability after each row. This is a reasonable observation since products that are displayed in the same row are all observed at the same time, without the need for scrolling. In contrast, examining products of different rows requires the user to scroll through the SERP. To validate this observation, the right plot of figure **??** shows the autocorrelation after first differencing the learned

Fig. 4.8.: The CTR per position (left) and the position bias estimates (right) for the list-based layout, broken down by the device type

examination probabilities. The autocorrelation exhibits significant spikes for lags 3 and 6 at the 95% confidence level, indicating a periodic pattern occurring after every third position, that is, after each row. This validates the hypothesis that the examination probability varies stronger between adjacent products of different rows than between products of the same row.

Fig. 4.7.: The left plot shows the learned position biases on the Otto data for the first 48 positions of a grid layout shown on desktop devices. The right plot depicts the autocorrelation computed on the first differences of these examination probabilities

Figure **??** shows the click-through rate per position for the list-based layout of all three device types (left), along with their position bias estimates (right). While the CTR per position is not a good estimate of the position bias as mentioned in section 2.3, the examination probability poses an upper bound for the positional CTR [Guo20b]. Consequently, both figures are related, which makes their comparison interesting. Since the presented rankings are independent of the device type, the differences in the positional CTR per device can be attributed to the examination probability. Following this assumption, the positional CTR indicates a faster decay of the examination probability on mobile devices than on desktop computers and tablets, where tablet users are the most patient. This is also consistent with what was learned by the Bias Estimator, whose normalized estimates are presented in the right-hand plot of figure **??**. This again confirms the effectiveness of the presented approach in the presence of device- and layout-dependent position biases.

4.3 Online A/B Test

4.3.1 Benchmark Description

The benchmark model is a manually tuned ranking function, which has been optimized for years by domain experts. Despite being a manual approach, the status quo ranking is a challenging benchmark and has just recently beaten a LambdaMART candidate ranking model in an extensive A/B test. Similar to the ranking function proposed in this work, the benchmark ranker first determines the context of the search query, which is determined by the dominant product category of the retrieved product collection. Depending on this context, different weights for the features are applied. For example, for the context “Electronics & Digital”, the ranking function puts more emphasis on a feature like freshness due to the short product lifecycles in this product category. For “Fashion & Sport” on the other hand, the price has a larger influence. Moreover, the revenue a product has generated in the past two weeks, the availability of a product and the BM25 score are used as features among many others. The

weights for these features have been optimized and fine-tuned throughout dozens of A/B tests and are constantly updated.

4.3.2 Experimental Setup

In the online A/B test, the effect of using the rankings generated by the neural ranking model is directly compared with the status quo ranking. To do so, static rankings for the 5,000 most issued queries are generated offline by the neural ranker. Users are randomly split into two groups, where a user of group A receives the status quo ranking when issuing one of the affected queries and a user of the test group B receives the static ranking generated by the proposed neural ranking model. This approach is chosen in order to neglect potential latency issues which might arise when dynamically computing the ranking for a new query. Since the goal of this thesis is to develop a competitive ranking model, the efficient serving of such a model is left for future work. Yet, the 5,000 identified queries make up for around 25% of the search traffic on otto.de and thus provide enough data to evaluate the quality of the generated rankings.

To validate the effect of the multi-task ranking approach on the business KPIs, two online A/B tests are conducted. For the first test, the ranking model is trained only on clicks and in the second test, the model is trained on both clicks and orders. In both cases, the GRU-DSSM is used as the encoder, AttnRank as scoring function and the JoE architecture is utilized to isolate the position bias. The only difference in the models is the MMoE layer, which reduces to a single hidden layer of fully connected units when using only clicks as the training target.

Several KPIs are used to evaluate the models. First, the effect on the click-through rate (CTR), which is defined as the percentage of query sessions, in which at least one product from the SERP is clicked, is measured. Like the ranking metrics, the click-through rate can be defined in terms of a position k, in which case it specifies the percentage of query sessions, in which at least one product from the first k positions is clicked. Second, the conversion rate (CvR) measures the percentage of customers who complete a specified customer journey. For a search engine, the journey starts with the customer issuing a search query and the journey ends with the customer purchasing a retrieved product. Moreover, the number of orders, the number of sold product units as well as the achieved revenue following a search query are determined.

4.3.3 Results

The online A/B test with the single-task model was conducted for one week. During that time, 1,173,201 query sessions were recorded for group A and 1,167,995 for group B. The results are demonstrated in table 4.7. Similar to the offline results of the ranking model trained only on clicks, the CTR increased significantly compared to the production ranker, while most of the order-related KPIs like CvR or revenue decreased. However, for these KPIs, no statistically significant change could be

Tab. 4.7.: Performance of the two tested ranking models compared to the status quo in the A/B test. An asterisk indicates statistical significance at the 95% confidence level

KPI	Single-task		Multi-task	
	status quo	test group	status quo	test group
CTR	57.76%	58.28% (+0.9%)*	58.50%	59.12% (+1.0%)*
CTR@3	18.81%	19.63% (+4.3%)*	19.73%	21.00% (+6.4%)*
CvR	4.83%	4.82% (-0.1%)	4.30%	4.31% (+0.3%)
Orders	59,174	58,859 (-0.5%)	176,270	177,386 (+0.6%)
Units	237,385	238,210 (+0.3%)	574,956	579,200 (+0.7%)
Revenue	13.213M	13.129M (-0.6%)	39.621M	39,692M (+0.2%)

measured since the decline is rather marginal. This is in contrast to the offline results, which show a strong decline in the order-related metrics for the single-task model compared to the status quo. Moreover, the decline can partly be attributed to the fact that the test group received around 0.4% fewer sessions. Therefore, the results for the single-task model are already promising, especially due to the substantial increase of the CTR in the top 3 positions, indicating that more relevant products have been ranked at the top of the SERP. The increase in CTR has led to about 40,000 more product views in absolute terms. This also positively impacted the total number of products sold, which rose by 0.3% from 237,385 in the status quo to 238,210 in the test group. Despite that, the revenue decreased slightly since the average price of purchased products decreased from 55.66€ in the status quo to 55.11€ in the test group. The decrease in the average order value results from the fact that the neural ranking model ranks cheaper products higher than the production ranker. The average price of the first 20 products in the SERP decreased by more than 8% from 232€ to 213€.

The A/B test for the multi-task ranking model has been conducted for three weeks. The runtime for this experiment was extended compared to the single-task model because we did not observe a decrease in revenue with the multi-task model, which is an essential criterion for the runtime of a test. Group A users have issued a query from the considered query set 4,103,119 times. The number of relevant queries issued in the test group totals 4,108,059. The results for the ranking model trained on both orders and clicks are also presented in table 4.7. Similar to the offline experiment, this model achieves an even larger improvement in the click-through rate than the single-task model. The overall CTR increased by 1.1% and the CTR@3 by 6.4% compared to the status quo, while the single-task model achieved improvements of "only" 0.9% and 4.3%, respectively. Moreover, unlike the single-task model, the neural ranker trained on clicks and orders improved upon the status quo in terms of the order-related KPIs. Thus, the multi-task model achieved better results than the benchmark for all relevant KPIs.

Tab. 4.8.: Micro-Conversions for the status quo and the test group. An asterisk indicates statistical significance at the 95% confidence level

Group	Conversion Rate			
	Query → SERP	**SERP → PDP**	**PDP → A2B**	**A2B → Order**
A	4,103,119	58.5%	14.2%	51.8%
B	4,108,059 (+0.1%)	59.1% (+1.1%)*	**13.7% (-3.1%)***	53.2% (+2.6%)*

However, the changes in the order-related KPIs are still not significant, despite the long runtime of the experiment. Also, the improvement of the conversion rate is substantially lower than the increase in the CTR, indicating a decline in the CvR somewhere within the funnel. To understand better, where in the journey customers are lost, the conversion of intermediate steps of the funnel, so-called micro-conversions, are analyzed. For example, the CvR (PDP → A2B) measures the percentage of customers who add a product after clicking on it to the basket. The distinct steps in the customer journey after issuing a search query are shown below:

Fig. 4.9.: Search Engine Funnel

Starting from the SERP, the user lands on the product detail page by clicking on a product. The next step is to add the product to the shopping basket. Finally, the journey ends with the order of the products in the basket. Table 4.8 shows the micro-conversions for all these steps for both the status quo and the test group, along with the relative change of each micro-conversion compared to the status quo. The conversion from a query to the SERP is given in absolute numbers since this step constitutes the starting point of the search journey and thus indicates how many customers of the respective groups entered the funnel. The conversion from the SERP to the product detail page (PDP) represents the click-through rate, which increased significantly for the test group, as already seen. However, the conversion from the PDP to an "add-to-basket" (A2B) decreased significantly by 3.1% in the test group, while the percentage of customers who ultimately purchase a product from the basket increased again.

To understand why so many more customers from the test group refrained from putting a product they had clicked on to the shopping basket, the products with the lowest micro-conversion from the detail page to an add-to-basket event are analyzed. First, many of the products exhibit high prices. However, as with the single-task ranker, the multi-task model on average ranks cheaper products higher on the SERP than the production ranker. An increase in the average price of the products in the SERP can consequently not be the reason for the decline in the analyzed micro-conversion. However, it also stood out that many of the analyzed products are not directly available, but have a high delivery time. This probably caused many customers not to order the products in question. When comparing the percentage

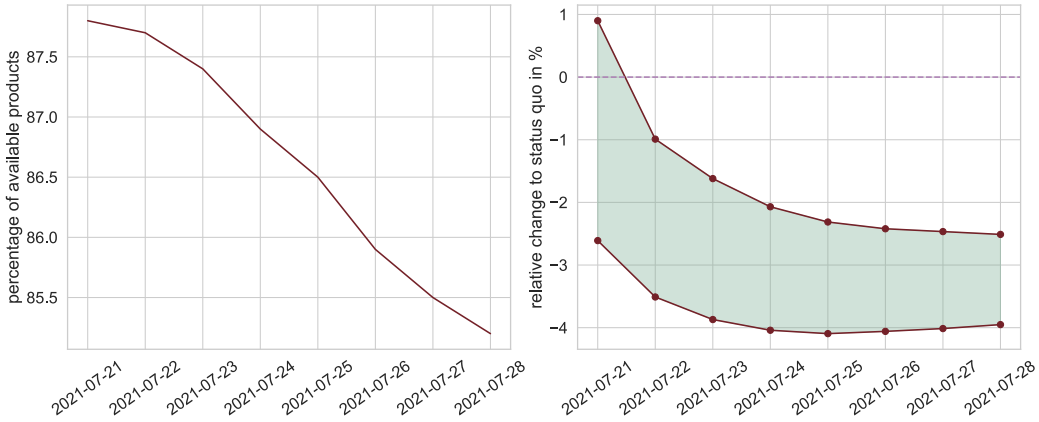


Fig. 4.10.: The percentage of clicked products, that are available (left) and the 95% confidence interval of the conversion rate from a PDP to an A2B (right)

of available products in both groups, a substantial decline in this figure is observed in the test group. Moreover, the percentage of clicked products that are available decreases in the test group with the duration of the experiment, as shown in the left plot of figure 4.10. A similar downward trend can be seen for the percentage of customers converting from the product detail page to the basket, as shown in the right plot of figure 4.10. These findings will be discussed more in-depth in the next chapter.

5 Conclusion

5.1 Discussion of Results

The goal of this thesis was to develop a Learning-to-Rank framework, which can use multiple sources of implicit feedback to learn the relevance of products without getting trapped in feedback loops due to the position bias. Furthermore, neural rankers should form the hypothesis space due to their ability to process raw textual information and their recent achievements in the LTR field. Throughout this work, several approaches have been identified and developed to fulfill these requirements.

The first outcome of this thesis is the encoder, whose main purpose is to encode queries and documents to compare their semantic distance in a low-dimensional space. The evaluation of the encoder is difficult without manual human assessment, since there are no ground-truth labels. Although the presented encoder architectures perform well on the product classification task, this must not necessarily mean that the learned embeddings accurately capture the semantic meaning of a text. For the manually inspected queries, the encoder managed to place similar products closer to the query than dissimilar ones. However, this evaluation should be repeated on a larger scale with several human assessors. Moreover, the example queries also revealed that the encoders do not always manage to place hard negatives far apart from the anchor in the embedding space. On the one hand side, this is probably because the sampling of negative examples is still not optimal. Although the chance of obtaining hard negative examples with batches as large as 1024 examples is reasonable, there is also a good chance of introducing false negatives when choosing a large batch size. On the other hand, due to the large amount of noise prevalent in click data, not all clicked products are relevant for the respective query. Together, these effects can significantly worsen the quality of the resulting embeddings.

To detect and isolate the effect of the position bias on the training target, this work has presented the JoE architecture. Although there are no ground-truth labels for the position bias either, we were able to evaluate JoE using simulated data, for which the true examination probabilities are known. By comparing the results of JoE to a strong baseline on this data, the effectiveness of this approach, especially in the presence of multiple layouts, could be demonstrated. Moreover, the position bias estimates on the real world data from the Otto online shop exhibit reasonable patterns. Not only do the JoE estimates capture the differences in the examination probabilities of the devices seen in the positional CTR, but it also detects the expected pattern on the grid layout, where the position bias declines stronger for positions corresponding to a row shift.

The resulting ranking model trained on both clicks and orders was then capable of achieving promising results on the validation set and managed to outperform the production ranker on all offline metrics. In the online A/B test, the proposed approach

significantly outperformed the status quo on the click-through rate, especially for top positions. For the order-related KPIs however, no significant improvement could be detected. Although the A/B test results of the multi-task model improved substantially compared to that of the single-task model, it was not able to significantly outperform the production ranker, which is not quite in line with the results observed offline, where large relative improvements were measured.

The reasons for this can be manifold. First, the experimental design of the A/B test does not benefit the neural ranking model. The use of a static ranking list has the drawback that changes in the input space do not affect the ranking after it has been deployed into the search system. Discounts on products, for example, change very frequently due to marketing events. However, these changes cannot be taken into account in the static list, while the status quo ranking updates such information continuously, giving it an unfair advantage. The severity of this problem has been demonstrated by the example of the delivery time. Although the ranking function learns that long delivery times harm the order probability of products (see figure 4.3d), the average fraction of available products decreased from 91.6% in the status quo ranking to 88.2% in the test group. One reason for this is that, especially on highly frequented queries, delivery times of products change very often since they become unavailable. As a result, the longer the static ranking is online, the more products at the top of the SERP exhibit long delivery times. This caused a significant decline in the conversion rate from the product's detail page to the basket in the test group, because customers did not accept the long delivery times of the affected products, despite being interested in them. However, since the change in the overall conversion is still positive, it can be assumed that the proposed method can achieve a significant uplift in the total conversion when this issue is fixed.

Another reason why the offline results have not been accomplished in the online test is the sampling of the 5,000 most frequently issued queries, which do not adequately represent the population of queries and products. The status quo heavily relies on the revenue achieved by a product in the past two weeks. This makes the production ranker a reliable model for queries with high traffic, but on long-tail queries, most of the products do not even sell once within a two week observation period. This might lead to poor rankings of the production ranker for queries with low traffic, while we assume the proposed neural ranker to work well on all queries alike. Due to the time restriction, we were not able to validate this assumption online. However, we observed an increase of the offline metrics for the production ranker when reducing the validation set to the queries used in the online A/B test, while the metrics for the neural ranker remained almost unchanged. This suggests that the neural ranker can further improve upon the benchmark in terms of the online metrics when evaluated on a larger query set.

5.2 Future Work

To further improve the quality of the neural ranking model and to mitigate the issues mentioned above, some areas of future work have been identified throughout this project. The most considerable improvements can probably be achieved by dynamically computing the rankings after the user issued the query, instead of precomputing a ranking for some subset of queries. In the future, we will therefore focus on serving the model live in the Otto shop. This might require replacing some model components like the MHSA layers with more efficient architectures, as well as developing techniques to precompute and cache as much information as possible. An alternative to serving the model live would be to improve the current setup to automatically update the static ranking file daily or even multiple times a day, so that changes in the feature space influence the displayed rankings regularly.

Serving the model online would also enable the use of user-related information. With the current setup, these cannot be used as it would require generating a static ranking for every possible user. However, user-related information like a user's shopping history, age or gender can improve the quality of the rankings significantly because a product's relevance largely depends on the preferences of the user who issued the query. In future work, we will investigate the possibilities of leveraging such information. One such way is to encode the user information similar to the query and document texts and compare the resulting user embeddings with the query embedding. Hidasi and Karatzoglou (2018) for example use RNNs to generate user embeddings, which encode the purchase history and, therefore, the users' preferences.

Moreover, since a decline in the revenue was measured for the test group in the A/B test, despite an increase in the number of sold products, a way to optimize the ranking in terms of revenue should be developed in the future. Wu et al. (2018) for example multiply the predicted order probability of a product by its price to receive the expected GMV of a product, according to which they rank the products in the SERP. A similar strategy might also be applicable to the multi-task framework presented in this work. Alternatively, more expensive products can be weighted stronger in the loss function of the order prediction task. We will investigate the impact of such a technique in a future A/B test. Also, since order-related KPIs have not been improved significantly compared to the status quo, it might be reasonable to increase the weight of the order predictions in the ranking score. Optimally, the weights for both tasks are tuned in the context of multiple online A/B tests.

Lastly, the quality of the rankings might be further enhanced by improving the encoder. While the architectures themselves do not seem to influence the overall quality of the embeddings too much, we saw that the training strategy has a substantial influence. While the performance of the triplet loss is mediocre regardless of the batch size, the models based on the softmax loss constantly yield better results. Especially when using a large batch size and taking all other documents as negatives, this method achieves good results. It can be assumed that by addressing the abovementioned issues, the overall quality of the embeddings can be improved significantly. Initial

ideas are to use the aggregated clicks on a product for a specific query as a weight for sampling positive examples for that query or to incorporate product properties like the PBK in a sampling logic. In addition to the new sampling strategy, a way to adequately evaluate the resulting embeddings must also be developed in the future.

5.3 Summary

To summarize, this thesis has successfully shown that implicit customer feedback found in clickthrough data provides valuable information that can be used to learn a ranking function. In the presence of multiple relevance signals that correspond to different business objectives, the approach presented in this work can even be used to learn a ranking that optimizes the various performance indicators of these objectives. In order to leverage the data, the position bias must be taken into account. We have shown that the presented JoE architecture is effective in estimating the examination probability of a product based on different sources of information, not only the position. On average, only 1 to 2 items from the first 10 positions of the status quo are also in the first 10 positions of the proposed ranking of the multi-task model. This validates that the presented method can effectively learn from implicit customer feedback without getting trapped in a feedback loop, in which the same products are ranked at the top over and over again.

The A/B test conducted as part of this work showed promising results, indicating the effectiveness of the presented methods. The multi-task ranker managed to improve all relevant KPIs compared to the status quo, even though no significant improvements in the order-related KPIs could be measured. We discussed several reasons for this and gave recommendations for future work that might lead to even more substantial improvements of the proposed neural ranker. All in all, the presented model has been developed in the complex environment of a large-scale e-commerce company and many of the technical challenges could not be attributed due to the time restriction. Therefore, the presented approach can be seen as a proof-of-concept, demonstrating the feasibility of applying complex neural networks to the ranking task, but needs further development to improve the ranking in a live environment.

Proof of Theorem 1

A

Theorem 1. *Given two losses $\mathcal{L}_1$ and $\mathcal{L}_2$, a learning algorithm minimizing a loss function of the form $\tilde{\mathcal{L}} = \lambda\mathcal{L}_1 + (1-\lambda)\mathcal{L}_2 + |\lambda\mathcal{L}_1 - (1-\lambda)\mathcal{L}_2|$ also minimizes the absolute difference between the weighted losses $\lambda\mathcal{L}_1$ and $(1-\lambda)\mathcal{L}_2$.*

Since the goal is to find a λ, which balances both losses and hence brings the absolute difference between them towards zero, the optimal λ can be derived as follows:

$$\lambda\mathcal{L}_1 - (1-\lambda)\mathcal{L}_2 = 0 \tag{A.1}$$

$$\lambda^* = \frac{\mathcal{L}_2}{\mathcal{L}_1 + \mathcal{L}_2}\,. \tag{A.2}$$

In order to minimize the absolute difference $|\cdot|$, $\tilde{\mathcal{L}}$ has to have a minimum at $\lambda = \frac{\mathcal{L}_2}{\mathcal{L}_1+\mathcal{L}_2}$.

Proof. For ease of notation, let a and b denote the loss of the first and second task respectively and $u = \lambda a - (1-\lambda)b$, then

$$\frac{\partial\tilde{\mathcal{L}}}{\partial\lambda} = a - b + \frac{u}{|u|}u' \tag{A.3}$$

$$= a - b + \frac{(a+b)u}{|u|} \overset{!}{=} 0 \tag{A.4}$$

Isolating the absolute value $|u|$ yields

$$|\lambda a - (1-\lambda)b| = \frac{(a+b)(\lambda a - (1-\lambda)b)}{b-a}\,. \tag{A.5}$$

Now, eq. (A.5) can be solved for λ by considering the two cases $u > 0$ and $u < 0$.

Case 1. *Proof for $\lambda a - (1-\lambda)b \geq 0$*

$$\lambda a - (1-\lambda)b = \frac{\lambda a^2 + 2\lambda ab - ab + \lambda b^2 - b^2}{b-a} \qquad |+b$$

$$\lambda(a+b) = \frac{(b-a)b}{b-a} + \frac{\lambda a^2 + 2\lambda ab - ab + \lambda b^2 - b^2}{b-a}$$

$$\lambda(a+b) = \frac{\lambda a^2 + 2\lambda ab + \lambda b^2 - 2ab}{b-a} \qquad |\cdot(b-a)$$

$$\lambda(a+b)(b-a) = \lambda(a+b)^2 - 2ab$$

$$\lambda(a+b)^2 - \lambda(b^2-a^2) = 2ab$$

$$2\lambda a^2 - 2\lambda ab = 2ab \qquad |:2a$$

$$\lambda = \frac{b}{a+b} = \frac{\mathcal{L}_2}{\mathcal{L}_1 + \mathcal{L}_2}$$

Case 2. *Proof for* $\lambda a - (1-\lambda)b < 0$

$$\begin{aligned}
\lambda a - (1-\lambda)b &= -\frac{\lambda a^2 + 2\lambda ab - ab + \lambda b^2 - b^2}{b-a} && | + b \\
\lambda(a+b) &= \frac{(b-a)b}{b-a} + \frac{\lambda a^2 + 2\lambda ab - ab + \lambda b^2 - b^2}{b-a} \\
\lambda(a+b) &= \frac{2b^2 - \lambda a^2 - 2\lambda ab - \lambda b^2}{b-a} && | \cdot (b-a) \\
\lambda b^2 - \lambda a^2 &= 2b^2 - \lambda a^2 - 2\lambda ab - \lambda b^2 \\
2\lambda b^2 &= 2b^2 - 2\lambda ab && | : 2b \\
\lambda(b+a) &= b \\
\lambda &= \frac{b}{a+b} = \frac{\mathcal{L}_2}{\mathcal{L}_1 + \mathcal{L}_2}
\end{aligned}$$

In both cases λ is identical to λ^* of equation (A.2). Moreover, it is easy to show that the extrema is a minimum using example data. Let $\mathcal{L}_1 = \mathcal{L}_2 = 0.5$, then the optimal λ is 0.5 and the first derivative of $\tilde{\mathcal{L}}$ reduces to:

$$\tilde{\mathcal{L}}'(\lambda) = \frac{\lambda - 0.5}{|\lambda - 0.5|} \tag{A.6}$$

Now it is easy to see that

- for $\lambda < 0.5$: $\tilde{\mathcal{L}}'(\lambda) < 0$, i.e. the function decreases
- for $\lambda > 0.5$: $\tilde{\mathcal{L}}'(\lambda) > 0$, i.e. the function increases

Hence, $\lambda = 0.5$ is the minimum point. □

Model Features

B

Tab. B.1.: Features used as Input for the Scoring Function f

Level	Feature	Description
Query	query_length	The number of tokens in the query
	token_sum	The sum of the term frequencies over the tokens in the query
	query_occurrences	The number of times the query was issued
Product	num_clicks	The number of clicks the product received over a time period of 3 weeks prior to the training set
	num_orders	The number of times the product was ordered over a time period of 3 weeks prior to the training set
	coec	The *clicks over expected clicks*
	avg_freshness	The number of days the product exists in the shop
	product_pd_RatingCount	The number of customer reviews
	product_pd_DiscountPercentage	The discount in %
	product_pd_AverageRating	The average score of all customer reviews ranging from 1 (bad) to 5 (excellent)
	price	The price of the product in €
	delivery_time	The number of days required to ship the product to the customer
	shipping_costs	The cost for shipping the product in €
Query-Product	tfidf_score	Sum of TF-IDF scores of all query tokens
	sms	The cosine similarity between product and query embeddings
	pbk_match	The predicted probability that a user searches for a product from the same product category as this product
	brand_match	Binary variable indicating whether the query contains the brand name of the product

Bibliography

[Aga19a] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. arXiv:1803.08375 [cs, stat]. Feb. 2019 (cit. on p. 27).

[Aga19b] Aman Agarwal, Kenta Takatsu, Ivan Zaitsev, and Thorsten Joachims. „A General Framework for Counterfactual Learning-to-Rank“. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris France: ACM, July 2019, pp. 5–14 (cit. on pp. 2, 8).

[Ai18a] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. „Learning a Deep Listwise Context Model for Ranking Refinement“. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (June 2018). arXiv: 1804.05936, pp. 135–144 (cit. on p. 27).

[Ai18b] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft. „Unbiased Learning to Rank with Unbiased Propensity Estimation“. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (June 2018). arXiv: 1804.05938, pp. 385–394 (cit. on pp. 3, 13, 35).

[Ai19] Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Michael Bendersky, and Marc Najork. „Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks“. In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. ICTIR '19. New York, NY, USA: Association for Computing Machinery, Sept. 2019, pp. 85–92 (cit. on pp. 2, 26, 33).

[Baj18] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. arXiv:1611.09268 [cs]. Oct. 2018 (cit. on pp. 8, 27).

[Bel19] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. *Seq2Slate: Re-ranking and Slate Optimization with RNNs*. arXiv:1810.02019 [cs, stat]. Mar. 2019 (cit. on p. 28).

[Beu18] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H. Chi. „Latent Cross: Making Use of Context in Recurrent Recommender Systems“. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. Marina Del Rey CA USA: ACM, Feb. 2018, pp. 46–54 (cit. on p. 29).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. arXiv:1607.06450 [cs, stat]. July 2016 (cit. on p. 29).

[BN06] Christopher M. Bishop and Nasser M. Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006 (cit. on p. 27).

[BR99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval*. Vol. 463. ACM press New York, 1999 (cit. on pp. 1, 2).

[Bre18] Eliot P Brenner. „End-to-End Neural Ranking for eCommerce Product Search“. In: *arXiv preprint arXiv:1806.07296* (2018), p. 10 (cit. on p. 6).

[BRL06] Christopher Burges, Robert Ragno, and Quoc Le. „Learning to rank with nonsmooth cost functions“. In: *Advances in neural information processing systems* 19 (2006) (cit. on p. 6).

[Bru19a] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. „An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance“. In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. Santa Clara CA USA: ACM, Sept. 2019, pp. 75–78 (cit. on p. 33).

[Bru19b] Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and Marc Najork. „Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks“. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Paris France: ACM, July 2019, pp. 1241–1244 (cit. on pp. 5, 6).

[Bur05] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. „Learning to rank using gradient descent“. In: *Proceedings of the 22nd international conference on Machine learning - ICML '05*. Bonn, Germany: ACM Press, 2005, pp. 89–96 (cit. on p. 5).

[Bur10] Christopher JC Burges. „From ranknet to lambdarank to lambdamart: An overview“. In: *Learning* 11.23-581 (2010). Publisher: Citeseer, p. 81 (cit. on p. 6).

[Cam10] B. Barla Cambazoglu, Hugo Zaragoza, Olivier Chapelle, Jiang Chen, Ciya Liao, Zhaohui Zheng, and Jon Degenhardt. „Early exit optimizations for additive machine learned ranking systems“. In: *Proceedings of the third ACM international conference on Web search and data mining*. 2010, pp. 411–420 (cit. on p. 1).

[Cao07] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. „Learning to rank: from pairwise approach to listwise approach“. In: *Proceedings of the 24th international conference on Machine learning - ICML '07*. Corvalis, Oregon: ACM Press, 2007, pp. 129–136 (cit. on pp. 5, 32).

[Car97] Rich Caruana. „Multitask learning“. In: *Machine learning* 28.1 (1997). Publisher: Springer, pp. 41–75 (cit. on p. 37).

[CC11] Olivier Chapelle and Yi Chang. „Yahoo! Learning to Rank Challenge Overview“. In: *Proceedings of the Learning to Rank Challenge*. ISSN: 1938-7228. PMLR, Jan. 2011, pp. 1–24 (cit. on pp. 2, 4, 27, 32).

[Cer18] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. *Universal Sentence Encoder*. arXiv:1803.11175 [cs]. Apr. 2018 (cit. on p. 7).

[Che20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. „A Simple Framework for Contrastive Learning of Visual Representations“. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, pp. 1597–1607 (cit. on pp. 21, 23, 43).

[Cho14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv:1406.1078 [cs, stat]. Sept. 2014 (cit. on p. 19).

[Cho15] François Chollet et al. *Keras*. 2015 (cit. on p. 17).

[CMR15] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. „Click models for web search“. In: *Synthesis lectures on information concepts, retrieval, and services* 7.3 (2015). Publisher: Morgan & Claypool Publishers, pp. 1–115 (cit. on p. 34).

[CZ09] Olivier Chapelle and Ya Zhang. „A dynamic bayesian network click model for web search ranking“. In: *Proceedings of the 18th international conference on World wide web*. WWW '09. New York, NY, USA: Association for Computing Machinery, Apr. 2009, pp. 1–10 (cit. on pp. 10, 35).

[Dev19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805 [cs]. May 2019 (cit. on pp. 8, 15).

[DP08] Georges E. Dupret and Benjamin Piwowarski. „A user browsing model to predict search engine click data from past observations.“ In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '08. New York, NY, USA: Association for Computing Machinery, July 2008, pp. 331–338 (cit. on pp. 2, 8, 10).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on p. 18).

[Goo21] Google. *Tensorflow Text: The wordpiece algorithm source code*. Feb. 2021. URL: `https://github.com/tensorflow/text` (visited on July 26, 2022) (cit. on pp. 16, 17).

[Goo22a] Google. *Organisation von Informationen – So funktioniert die Google-Suche*. 2022. URL: `https://www.google.com/intl/de/search/howsearchworks/how-search-works/organizing-information/` (visited on July 6, 2022) (cit. on p. 1).

[Goo22b] Google. *Subword tokenizers Tutorial*. May 2022. URL: `https://www.tensorflow.org/text/guide/subwords_tokenizer` (visited on July 26, 2022) (cit. on p. 16).

[Guo16] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. „A Deep Relevance Matching Model for Ad-hoc Retrieval“. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Oct. 2016). arXiv: 1711.08611, pp. 55–64 (cit. on pp. 7, 26).

[Guo20a] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. „A Deep Look into neural ranking models for information retrieval“. In: *Information Processing & Management* 57.6 (Nov. 2020), p. 102067 (cit. on pp. 6, 25).

[Guo20b] Ruocheng Guo, Xiaoting Zhao, Adam Henderson, Liangjie Hong, and Huan Liu. „Debiasing Grid-based Product Search in E-commerce“. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 2852–2860 (cit. on pp. 36, 52).

[Hal19] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. „Applying Deep Learning to Airbnb Search“. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. New York, NY, USA: Association for Computing Machinery, July 2019, pp. 1927–1935 (cit. on p. 11).

[Hal20] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tyler Sax, Lanbo Zhang, Aamir Mansawala, Shulin Yang, Bradley Turnbull, and Junshuo Liao. „Improving Deep Learning For Airbnb Search“. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Feb. 2020) (cit. on p. 10).

[He15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs]. Dec. 2015 (cit. on p. 29).

[HK18] Balázs Hidasi and Alexandros Karatzoglou. „Recurrent Neural Networks with Top-k Gains for Session-based Recommendations“. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. arXiv:1706.03847 [cs]. Oct. 2018, pp. 843–852 (cit. on p. 59).

[Hu14] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. „Convolutional Neural Network Architectures for Matching Natural Language Sentences“. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014 (cit. on p. 7).

[Hu19] Ziniu Hu, Yang Wang, Qu Peng, and Hang Li. „Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm“. In: *arXiv:1809.05818 [cs]* (Feb. 2019). arXiv: 1809.05818 (cit. on pp. 4, 5, 35).

[Hua13] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. „Learning deep structured semantic models for web search using clickthrough data“. In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2013, pp. 2333–2338 (cit. on pp. 7, 15, 17).

[IS15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167 [cs]. Mar. 2015 (cit. on p. 25).

[Joa02] Thorsten Joachims. „Optimizing search engines using clickthrough data“. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '02. New York, NY, USA: Association for Computing Machinery, July 2002, pp. 133–142 (cit. on p. 8).

[JSS17] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. „Unbiased Learning-to-Rank with Biased Feedback“. In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. Cambridge United Kingdom: ACM, Feb. 2017, pp. 781–789 (cit. on pp. 9, 10, 33, 49, 50).

[Kud18] Taku Kudo. *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*. arXiv:1804.10959 [cs]. Apr. 2018 (cit. on p. 17).

[Li21] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. „Embedding-based Product Retrieval in Taobao Search“. In: *arXiv preprint arXiv:2106.09297* (2021) (cit. on p. 14).

[Lin17] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. „Model ensemble for click prediction in bing search ads“. In: *Proceedings of the 26th international conference on world wide web companion*. 2017, pp. 689–698 (cit. on p. 11).

[Liu09] Tie-Yan Liu. „Learning to Rank for Information Retrieval“. In: *Foundations and Trends in Information Retrieval* 3.3 (Mar. 2009), pp. 225–331 (cit. on pp. 1, 4, 32).

[Liu22] Weiwen Liu, Yunjia Xi, Jiarui Qin, Fei Sun, Bo Chen, Weinan Zhang, Rui Zhang, and Ruiming Tang. *Neural Re-ranking in Multi-stage Recommender Systems: A Review*. arXiv:2202.06602 [cs]. Apr. 2022 (cit. on p. 13).

[LL17] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. arXiv:1705.07874 [cs, stat]. Nov. 2017 (cit. on p. 46).

[LÖ09] Ling Liu and M. Tamer Özsu, eds. *Encyclopedia of database systems*. Springer reference. New York: Springer, 2009 (cit. on p. 31).

[LSR14] Damien Lefortier, Pavel Serdyukov, and Maarten de Rijke. „Online Exploration for Detecting Shifts in Fresh Intent“. In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. CIKM '14. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 589–598 (cit. on p. 2).

[Ma18] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. „Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts“. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. London United Kingdom: ACM, July 2018, pp. 1930–1939 (cit. on pp. 37, 38, 49).

[MC18] Bhaskar Mitra and Nick Craswell. „An introduction to neural information retrieval“. In: *Foundations and Trends® in Information Retrieval* 13.1 (2018). Publisher: Now Publishers, Inc., pp. 1–126 (cit. on p. 7).

[Moe03] Wendy W. Moe. „Buying, Searching, or Browsing: Differentiating Between Online Shoppers Using In-Store Navigational Clickstream“. In: *Journal of Consumer Psychology*. Consumers in Cyberspace 13.1 (Jan. 2003), pp. 29–39 (cit. on p. 11).

[NC20] Rodrigo Nogueira and Kyunghyun Cho. *Passage Re-ranking with BERT*. arXiv:1901.04085 [cs]. Apr. 2020 (cit. on p. 7).

[Nig19] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian (Allen) Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. „Semantic Product Search“. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. New York, NY, USA: Association for Computing Machinery, July 2019, pp. 2876–2885 (cit. on pp. 7, 12, 13, 36).

[OR20] Harrie Oosterhuis and Maarten de Rijke. „Policy-Aware Unbiased Learning to Rank for Top-k Rankings“. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (July 2020). arXiv: 2005.09035, pp. 489–498 (cit. on p. 9).

[Ott22] Otto. *OTTO (GmbH & Co KG) – Auf einen Blick*. Mar. 2022. URL: `https://www.otto.de/unternehmen/de/wer-wir-sind/auf-einen-blick` (visited on Aug. 22, 2022) (cit. on p. 41).

[Pag99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank citation ranking: Bringing order to the web*. Tech. rep. Stanford InfoLab, 1999 (cit. on p. 4).

[Pal16] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. „Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval“. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.4 (Apr. 2016). arXiv:1502.06922 [cs], pp. 694–707 (cit. on p. 7).

[Pan16] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. *Text Matching as Image Recognition*. arXiv:1602.06359 [cs]. Feb. 2016 (cit. on pp. 7, 25).

[Pas20] Rama Kumar Pasumarthi, Honglei Zhuang, Xuanhui Wang, Mike Bendersky, and Marc Najork. „Permutation Equivariant Document Interaction Network for Neural Learning to Rank“. In: *Proceedings of the 2020 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR 2020)*. 2020 (cit. on pp. 28, 29).

[Pia14] Steven T. Piantadosi. „Zipf's word frequency law in natural language: A critical review and future directions“. In: *Psychonomic bulletin & review* 21.5 (Oct. 2014), pp. 1112–1130 (cit. on p. 15).

[Rob21] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. *Contrastive Learning with Hard Negative Samples*. arXiv:2010.04592 [cs, stat]. Jan. 2021 (cit. on p. 21).

[RU11] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011 (cit. on p. 4).

[Sar20] Fatemeh Sarvi, Nikos Voskarides, Lois Mooiman, Sebastian Schelter, and Maarten de Rijke. „A Comparison of Supervised Learning to Match Methods for Product Search“. In: *arXiv:2007.10296 [cs]* (July 2020). arXiv: 2007.10296 (cit. on p. 14).

[SC16] Daria Sorokina and Erick Cantu-Paz. „Amazon Search: The Joy of Ranking Products“. In: *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. SIGIR '16. New York, NY, USA: Association for Computing Machinery, July 2016, pp. 459–460 (cit. on p. 11).

[SGR17] Ayan Sinha, David F. Gleich, and Karthik Ramani. *Deconvolving Feedback Loops in Recommender Systems*. arXiv:1703.01049 [cs]. Mar. 2017 (cit. on p. 3).

[Sha16] Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and Jc Mao. „Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features“. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM, Aug. 2016, pp. 255–262 (cit. on pp. 26, 27).

[She14] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. „A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval“. In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. Shanghai China: ACM, Nov. 2014, pp. 101–110 (cit. on p. 7).

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. „FaceNet: A unified embedding for face recognition and clustering“. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 815–823 (cit. on p. 20).

[Soh16] Kihyuk Sohn. „Improved Deep Metric Learning with Multi-class N-pair Loss Objective“. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016 (cit. on pp. 21, 23).

[Son21] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. *Fast WordPiece Tokenization*. arXiv:2012.15524 [cs]. Oct. 2021 (cit. on p. 17).

[Sri14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. „Dropout: a simple way to prevent neural networks from overfitting“. In: *The journal of machine learning research* 15.1 (2014). Publisher: JMLR. org, pp. 1929–1958 (cit. on p. 25).

[SSZ17] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. „On Application of Learning to Rank for E-Commerce Search“. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Aug. 2017). arXiv: 1903.04263, pp. 475–484 (cit. on pp. 11, 26).

[Tra21] Mohamed Trabelsi, Zhiyu Chen, Brian D. Davison, and Jeff Heflin. „Neural ranking models for document retrieval“. In: *Information Retrieval Journal* 24.6 (Dec. 2021), pp. 400–444 (cit. on p. 26).

[Vas17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. „Attention is All you Need“. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017 (cit. on pp. 7, 28–30, 45).

[VRK16] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. „Learning Latent Vector Spaces for Product Search“. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 165–174 (cit. on p. 14).

[Wan17] Feng Wang, Xiang Xiang, Jian Cheng, and Alan L. Yuille. „NormFace: L2 Hypersphere Embedding for Face Verification“. In: *Proceedings of the 25th ACM international conference on Multimedia*. arXiv:1704.06369 [cs]. Oct. 2017, pp. 1041–1049 (cit. on p. 23).

[Wan18] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. „Position Bias Estimation for Unbiased Learning to Rank in Personal Search“. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. WSDM '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 610–618 (cit. on pp. 34, 35).

[Wu16] Yonghui Wu, Mike Schuster, Zhifeng Chen, et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv:1609.08144 [cs]. Oct. 2016 (cit. on pp. 15, 19).

[Wu18a] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. „Turning Clicks into Purchases: Revenue Optimization for Product Search in E-Commerce“. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. Ann Arbor MI USA: ACM, June 2018, pp. 365–374 (cit. on pp. 11, 12, 37, 59).

[Wu18b] Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. *Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination*. arXiv:1805.01978 [cs]. May 2018 (cit. on p. 21).

[Xio17] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. „End-to-end neural ad-hoc ranking with kernel pooling“. In: *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*. 2017, pp. 55–64 (cit. on p. 22).

[Yao21] Shaowei Yao, Jiwei Tan, Xi Chen, Keping Yang, Rong Xiao, Hongbo Deng, and Xiaojun Wan. „Learning a Product Relevance Model from Click-Through Data in E-Commerce“. In: *Proceedings of the Web Conference 2021*. Ljubljana Slovenia: ACM, Apr. 2021, pp. 2890–2899 (cit. on pp. 6, 8).

[Zho16] Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. *Minimal Gated Unit for Recurrent Neural Networks*. arXiv:1603.09420 [cs]. Mar. 2016 (cit. on p. 19).

[Zhu20] Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. „Feature Transformation for Neural Ranking Models“. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Virtual Event China: ACM, July 2020, pp. 1649–1652 (cit. on p. 27).

[ZJ07] Wei Vivian Zhang and Rosie Jones. „Comparing click logs and editorial labels for training query rewriting“. In: *WWW 2007 Workshop on Query Log Analysis: Social And Technological Challenges*. 2007 (cit. on p. 10).

[ZK20] Xiaofeng Zhu and Diego Klabjan. „Listwise Learning to Rank by Exploring Unique Ratings“. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. arXiv:2001.01828 [cs, stat]. Jan. 2020, pp. 798–806 (cit. on p. 6).

Eidesstattliche Erklärung

Ich versichere, dass ich diese Master-Arbeit – bei einer Gruppenarbeit den entsprechend gekennzeichneten Teil der Master-Arbeit – selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass alle Stellen der Arbeit, die wortwörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht habe. Die schriftliche sowie die elektronische Fassung der Arbeit stimmen inhaltlich überein.

Hamburg, 06.09.2022

Laurin Luttmann