Matching Cross Network for Learning to Rank in Personal Search

Zhen Qin, Zhongliang Li, Michael Bendersky, Donald Metzler
Google LLC, Mountain View, CA, USA
{zhenqin,lzl,bemike,metzler}@google.com

ABSTRACT

Recent neural ranking algorithms focus on learning semantic matching between query and document terms. However, practical learning to rank systems typically rely on a wide range of side information beyond query and document textual features, like location, user context, etc. It is common practice to concatenate all of these features and rely on deep models to learn a complex representation.

We study how to effectively and efficiently combine textual information from queries and documents with other useful but less prominent side information for learning to rank. We conduct synthetic experiments to show that: 1) neural networks are inefficient at learning the interaction between two prominent features (e.g., query and document embedding features) in the presence of other less prominent features; 2) direct application of a state-of-art method for higher-order feature generation is also inefficient.

Based on the above observations, we propose a simple but effective matching cross network (MCN) method for learning to rank with side information. MCN conducts an element-wise multiplication matching of query and document embeddings and leverages a technique called latent cross to effectively learn the interaction between matching output and all side information. The approach is easy to implement, and adds minimal parameters and latency overhead to standard neural ranking architectures.

We conduct extensive experiments using two of the world's largest personal search engines, Gmail and Google Drive search, and show that each proposed component adds meaningful gains against a strong production baseline with minimal latency overhead, thereby demonstrating the practical effectiveness and efficiency of the proposed approach.

CCS CONCEPTS

• **Information systems → Information retrieval**.

KEYWORDS

neural networks, learning to rank, embeddings

ACM Reference Format:
Zhen Qin, Zhongliang Li, Michael Bendersky, Donald Metzler. 2020. Matching Cross Network for Learning to Rank in Personal Search. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3366423.3380046

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380046

1 INTRODUCTION

Neural ranking methods are commonly used today in a variety of information retrieval tasks. Most existing methods focus on semantic matching between query and documents using distributional representations learned from large amounts of data [16].

However, in addition to the query and document content, modern search engines leverage a variety of side information to improve their ranking performance. These features may not be as prominent as the content features, but can still boost performance if used appropriately. One example is situational context [30], such as location, platform (desktop vs phone), and time of the day, which are query and document independent. Another example is user-specific features, such as demographics and historical preferences, which are crucial for personalized search experiences [11]. How to effectively and efficiently combine all of this available side information is an important but not well explored problem.

Leveraging side information is especially important in the *personal search* setting, where users only have access to their own private documents (e.g., emails). Therefore the amount of training data available for each user is quite limited [3]. As a result, user and context features take on extra importance. For example, if a user searches "receipt" at an airport, it is more likely that she is looking for flight ticket receipts over other types of receipts.

A common practice for utilizing side information is concatenating all the features and relying on a deep model to learn the complex interactions between the features [30]. However, standard deep learning approaches are *inefficient* at learning complex representations from data [4] as they typically require both large amounts of data and large models. In practical large-scale applications, large amounts of high quality labeled data is often difficult to come by and large models can be a concern for latency-sensitive user-facing products. In this work, we study how to both effectively and efficiently combine textual information from queries and documents with side information. We begin with an empirical analysis of the conventional feed-forward network and a state-of-art feature interaction learning framework called latent cross [4]. We demonstrate that both are inefficient at capturing the interactions of two prominent features when other less prominent features exist. This finding is particularly relevant to information retrieval tasks where it is often the case that two prominent features (i.e., queries and documents) exist alongside other side information.

Motivated by these findings, we introduce a simple but effective approach called *Matching Cross Network* (MCN), which takes advantage of the dominance of query-document matching in information

The first two authors contributed equally.

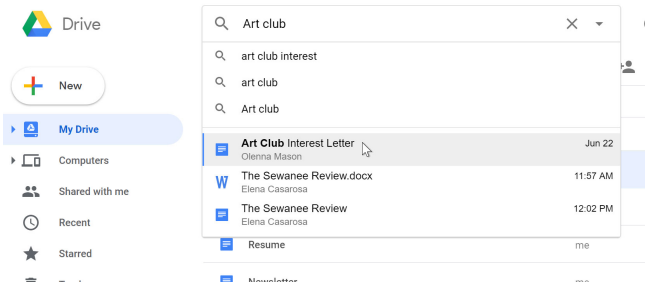

Figure 1: Google Drive "search-as-you-type" feature.

retrieval tasks. MCN performs an early query and document matching, then uses the latent cross idea to learn the interaction between the matching output and the additional side information. Through a rigorous ablation study on two large-scale personal search engines, GMail search and Google Drive search, we show that 1) early query-document matching is extremely useful, 2) though latent cross fails when used naively, it provides substantial benefits when combined with early matching.

The proposed MCN method is easy to implement, adds minimal parameter and latency overhead to standard neural ranking architectures, and can be efficiently trained over large-scale datasets. This makes the approach easy to integrate into existing systems. For example, MCN is currently deployed by Google Drive's "search-as-you-type" feature (see Figure 1) as it provided significant quality wins with minimal latency overhead and negligible added system complexity.

The remainder of this paper is organized as follows. We start by describing related work in Section 2. To motivate our work, Section 3 shows the inefficiencies of neural networks when two prominent features exist in the data alongside other side information. Sections 4 and 5 introduce our proposed matching cross network (MCN) technique. In Section 6 we verify the effectiveness of MCN on two of the world's largest personal search datasets through (a) a rigorous offline evaluation and (b) a large-scale online experiment. We conclude the paper in Section 7.

2 RELATED WORK

Neural ranking models have become popular in the information retrieval community. Most existing work focuses on semantic matching between queries and documents [23, 25, 28, 31]. For instance, the DUET model [17] considers local exact text matching combined with semantic embeddings. This architecture introduces considerable overhead with term matching matrix and Convolutional Neural Network (CNN) modules. [18] first builds a matching matrix between two texts, then treats the matrix as an image and uses CNNs for feature extraction. [6] explores CNNs and kernel pooling for soft matching between query and document n-grams. The Deep Semantic Similarity Model [12] uses Siamese networks to learn embeddings for matching. [20] focuses on social media search with small amounts of context. [31] tries to match query with multiple fields of the document, including document title and body. [19] explores the idea of probabilistic word embedding (PWE) for query and document matching, but found PWE is in general difficult to use and train due to the large amount of parameters. A recent survey on neural search can be found at [16]. Most prior work typically does not consider the rich side information available in practice. Our work demonstrates the importance of early matching for learning to rank when side information is available. Our proposed framework has a general query and document matching module where existing work can easily be leveraged.

On the other hand, large-scale search systems usually leverage efficient query-document matching [10] then re-rank top candidates with rich side information [29] to further boost performance. A common practice is to concatenate all features in a simple feed-forward architecture and rely on deep models to learn complex representations [2, 7, 14, 30]. Some recent work demonstrated it is inefficient for standard deep models to learn effective representations from data in the information retrieval domain. [22] tries to incorporate query clusters as a feature, but could not get any performance gain with concatenation. They thus design a multi-task objective to use query cluster prediction as an auxiliary task to guide the representation learning. Our work also tries to address the limitations of standard neural architectures for search ranking problems, with an emphasis on effectiveness, efficiency, and simplicity.

Side information, especially contextual features, are extensively explored in recommender systems research since such information is critical for making high quality recommendations. [24] directly feeds contextual information into a Recurrent Neural Network (RNN) for session-based recommendations. [32] multiplicatively incorporated temporal information into a long short-term memory (LSTM) model for better sequential recommendation. The recent work of latent cross [4] describes a simple technique to efficiently learn high-order feature interactions, such as the platform (mobile vs desktop) in Youtube video recommendation. Our work applies this state-of-art idea to learn the interaction between textual information and side information in search ranking.

3 MODELING MOTIVATION

We begin by demonstrating the limitations of standard feed-forward networks and latent cross [4] using a synthetic dataset that is meant to mimic typical search ranking problems.

Each example in the dataset consists of three 20-dimensional features $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$. Each feature value is initialized from a uniform distribution between -1 and 1. The label for each example is generated by *only* using information from $\mathbf{f}_1$ and $\mathbf{f}_2$ as

$$y = \begin{cases} 1, & \text{if } \cos(\mathbf{f}_1, \mathbf{f}_2) > 0 \\ 0, & \text{otherwise,} \end{cases}$$

where $\cos(\cdot)$ is the cosine similarity. This dataset imitates common search scenarios, with $\mathbf{f}_1$ and $\mathbf{f}_2$ being akin to query and document features. Three datasets are generated for training, validation, and test purposes, each consisting of 1,000 samples.

Three different model architectures are evaluated: 1) Concatenation of all features. 2) Using $\mathbf{f}_1$ as the latent cross with other features (i.e., element-wise multiplication between $\mathbf{f}_1$ and the output of deep layers using $\mathbf{f}_2$ and $\mathbf{f}_3$ as inputs, see more details in Section 5.2. 3) Conducting element-wise multiplication between $\mathbf{f}_1$ and $\mathbf{f}_2$ first, then concatenating the output with $\mathbf{f}_3$. All models use three hidden

layers. We use cross entropy loss and Adagrad [9] optimizer for training. We use different hidden layer widths to see the model's efficiency for modeling the data, similar to [4]. For each experiment, we pick the checkpoint with highest validation accuracy.

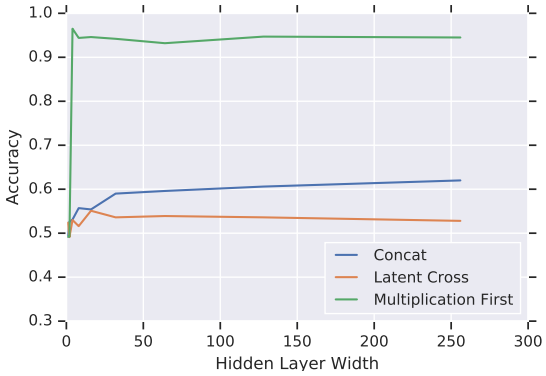

Figure 2: Simple feature concatenation and using latent cross are both inefficient at learning the cosine similarity.

The results on the test data are shown in Figure 2. We observe that concatenation in feed-forward network and naively applying latent cross are inefficient at modeling this seemingly simple synthetic dataset since the performance increases slowly with more hidden units. Our hypothesis of the failure of directly applying latent cross is, $\mathbf{f}_1$ and $\mathbf{f}_2$ are too separated in the model architecture so their interactions are difficult to learn. On the other hand, an early matching of two prominent features makes learning much easier - approach 3 serves as a sanity check and shows the dataset is learnable.

Note that the goal here is to simply illustrate a fundamental limitation of standard deep learning architectures on a dataset with two prominent features, similar to common search ranking settings. Our new model, which is motivated by these findings, will be verified on real-world search datasets in Section 6.

4 MODELING PRELIMINARIES

Suppose we have a set of queries $Q = \{q_i\}_{i=1}^{|Q|}$. Each query q_i is associated with a set of documents $\mathbf{d}_i = \{d_{i,1}, d_{i,2}, ..., d_{i,n_i}\}$ where n_i is the total number of documents corresponding to q_i. Let $x_{i,k}$ denote the training features for the query-document pair $\langle q_i, d_{i,k}\rangle$. Each query-document pair $\langle q_i, d_{i,k}\rangle$ is associated with a label $y_{i,k}$ that denotes the level of relevance for the pair. Typically, for click-through data, $y_{i,k}$ ($k \in [n_i]$) is a binary indicator of whether the user clicked document $d_{i,k}$. The learning-to-rank problem is to learn a function f that takes $x_{i,k}$ as input and outputs a relevance score between q_i and $d_{i,k}$ to optimize a predefined ranking objective

$$\max_f \frac{1}{|Q|}\sum_{i=1}^{|Q|} m\left(\{y_{i,k}, f(x_{i,k})\}_{k=1}^{n_i}\right) \tag{1}$$

where the metric $m(\cdot)$ is an evaluation metric like NDCG [5]. Directly optimizing the above objective is difficult because the metric $m(\cdot)$ is not smooth. Typical learning-to-rank algorithms use smooth surrogate objectives such as logistic or hinge loss [15] that are easier to optimize. Doing so converts (1) into the following minimization problem.

$$\min_f \bar{\ell} := \frac{1}{|Q|}\sum_{i=1}^{|Q|} \ell\left(\{y_{i,k}, f(x_{i,k})\}_{k=1}^{n_i}\right) \tag{2}$$

where $\ell(\cdot)$ is the loss function that can be pointwise, pairwise, or listwise [15].

5 METHODS

In this section, we describe the Matching Cross Network (MCN) model.

5.1 Embedding Features Matching

We use embeddings to represent query and document textual features, which is common in neural learning-to-rank systems [16, 17, 22]. For simplicity, we assume query and document terms use the same dictionary and embedding lookup table (so they also have the same embedding dimension), which reduces the number of model parameters. Query and document term embeddings are aggregated using the average word embedding (or average pooling) to represent queries and documents, which is a common practice [16].

Motivated by the synthetic experiments, we advocate the early matching of queries and documents before feeding them into deep architectures. We propose to use a simple matching between query and document embeddings as

$$f_{\text{matching}} = e_{\text{query}} \odot e_{\text{document}},$$

where $\odot$ is the element-wise multiplication operator. Note that more complex existing semantic matching methods [16] might also be used here. We find that a direct matching function via multiplication is simple to implement, adds minimal overhead to training time, model size, and inference latency. Also, using a shared embedding dictionary for queries and documents incorporates some of the textual matching ideas from DUET [17] (i.e., if the embeddings of queries and documents are the same, the output score will be high due to element-wise multiplication). We will show in experiments that this simple idea brings immediate performance gains over a well-tuned neural network that concatenates features. We also empirically show that several more complex matching functions do not provide significant extra gains.

5.2 Latent Cross

Latent Cross is a recent technique deployed in Youtube recommendation systems [4]. It feeds certain features into a separate shortcut and allows for more efficient higher order feature interaction learning. For example, in recommendation systems, it makes sense to implicitly generate feature interactions between user type and other item features.

$$f_{\text{cross}} = (1 + w_{\text{latent}}) \odot h_{\text{out}},$$

where w_{latent} is the embedding for context or latent features such as user type and platform, and h_{out} is the output of the final hidden layer of other features. In the context of search ranking, the latent cross attempts to generate feature interactions between queries

and other features. However, as we will show in the experimental evaluation section, this naive application of latent cross does not work well in practice.

5.3 Matching Cross Network

The general architecture of MCN is shown in Figure 3. MCN has two major components that combines the ideas from previous sections: 1) the early matching of query embeddings and document embeddings through a simple element-wise multiplication. 2) the matching output is used as latent cross features to facilitate the learning of its interactions with the additional side information.

The overall MCN architecture adds a negligible number of parameters as well as almost no additional latency compared with more basic architectures since the added operations are simply multiplications and no new features are introduced. The changes do not require any infrastructure changes (e.g. making some new features available or adding support for a new operator) in real-world serving systems, so it can be easily deployed and tested.

MCN may be easily extendable by treating it as the composition of a *matching* component that much of the neural ranking research has been focusing on [16] (though complexity and latency should be considered in practice), and a regular *deep* component where general learning-to-rank ideas [15] can be applied.

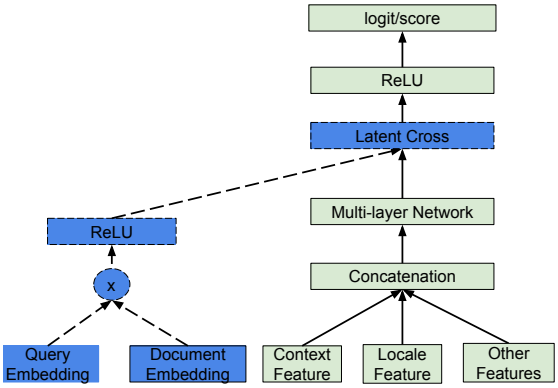

Figure 3: Diagram of the Matching Cross Network.

6 EXPERIMENTS

In this section, we first describe the data and the features used in the experiments; we then present the evaluation setup and metrics. Finally, we explore both the offline and the online experimental results using two of the world's largest personal search engines.

6.1 Data

Our experiments use the click-through data from Google Gmail and Drive, which we refer to as GMail and Drive for simplicity in the remainder of this section. For offline model training and evaluation, we collect GMail and Drive search logs between Aug. and Dec. 2018, resulting in hundreds of millions of queries with clicks. Among all the queries, 80% are used for training, 10% are used for validation and parameter tuning, and the remaining 10% are used for testing. All queries in the development set are issued strictly later than all queries in the training set, and all queries in the testing set are issued strictly later than all queries in the development (validation) set. On average, each GMail query has six candidate documents and each Drive query has five candidate documents based on their respective search interfaces.

To preserve user privacy, we only use frequent word and character n-grams that occur across multiple users to represent queries and documents and feed them into embedding layers. The n-gram embedding lookup table is randomly initialized since we have a large amount of data. The dataset also contains a rich set of side information, including situational features (e.g. time of the day), user features (e.g. user's previous click behavior), and non-textual document attributes (e.g. document age). As is common in personal search [3, 26, 30], we use click data as labels to learn and evaluate our proposed ranking models offline. Furthermore, we apply position bias correction techniques [26] during training to mitigate label bias from click data.

6.2 Experimental Setup and Metrics

We describe both offline and online metrics that will be used.

Offline metrics. We use weighted mean reciprocal rank (WMRR) as our primary metric offline, since it has been found to be predictive of online gains in prior work [26]. WMRR is calculated as follows:

$$\text{WMRR} = \frac{1}{\sum_{i=1}^{N} w_i} \cdot \sum_{i=1}^{N} w_i \frac{1}{\text{rank}_i}, \tag{3}$$

where w_i denotes the bias correction weight, which is inversely proportional to the probability of observing a click at the clicked position. The weights are set using methods described in [26] and the same set of weights are used for all models. N denotes the size of the evaluation set. rank_i denotes the rank position of the clicked document for the i-th query in the evaluation set. Higher WMRR numbers indicates the better model performance.

Online metrics. We track several metrics during user-facing online experiments. One key online metric is Click-Through Rate (CTR), measuring the percentage of search sessions during which users click a shown result. We also track a ranking-focused metric, the average clicked position (ACP) of the clicked results as

$$\text{ACP} = \frac{1}{M} \sum_{i=1}^{M} \text{rank}_i, \tag{4}$$

where M denotes the number of clicked sessions. Note that a lower ACP number indicates better performance. An ideal ranking model should increase CTR (so users click more) and decrease ACP (the clicked document is ranked higher). We also report additional user-facing metrics that focus on user search experience.

6.3 Offline Experimental Results

We compare the following models, where all the hyper-parameters are chosen via proper tuning on the condition that each model is trained and evaluated independently over the datasets. All the neural models are implemented using the Tensorflow toolkit [1] and trained with mini-batch stochastic gradient descent.

(1) **Baseline**. The baseline model is the concatenation model. All the features including query and document embeddings

are concatenated before being fed into deep layers. This is similar to the approach described by Zamani et al. [30]. The model has three fully-connected hidden layers. The number of neurons in these layers vary from 128 to 512 and the rectifiers are all ReLUs. All other models are compared against this baseline model.

(2) **Decision tree (DT)** is a popular technique used for search ranking, especially prior to neural ranking approaches. The DT model is an in-house implementation based on LambdaMART [27] with WMRR as the metric to compute the lambda gradient [8]. The final number of leaves across all trees is 1000. Note that DT is not a neural approach so it does not use embedding-based representations for queries and documents. Thus, the query and document embedding components are replaced with a BM25-based [21] matching score. All other features are fed into DT.

(3) **Convolutional Neural Network (CNN)** is a popular feature extractor for NLP tasks [13] that has also been explored for search tasks [17]. Similar to [17], we use CNN modules to extract features from query and document embeddings then concatenate them with other features before feeding them into the fully connected layers. Hyperparameters tuned include filter size, number of CNN layers, and stride size.

(4) **Latent Cross** This is a direct application of latent cross [4]. The model treats the query as a latent feature and naively applies latent cross to the output of a deep module with all other features.

(5) **Multiplication First**, which first gets the element-wise multiplication output of query and document embeddings then concatenates it with other features and feeds them into fully connected layers. In other words, this is MCN without latent cross.

(6) **MCN**. The proposed architecture, as shown in Figure 3.

(7) **Kernel MCN**. We also experimented with a straightforward extension of MCN by increasing its model capacity. Instead of an element-wise multiplication between query and document embedding as in Sec 5.1, we can use a linear projection matrix between them:

$$f_{\text{matching}} = e_{\text{query}} \cdot P \cdot e_{\text{document}},$$

which allows interactions between embedding dimensions. This model reduces to MCN when P is the identity matrix. We also tried more complex interaction operators but the results were similar so we omit them.

The offline experiment results are shown in Table 1. It should be noted that in large commercial search systems, an approach with an improvement around 0.1% over a large-scale evaluation data set is fairly significant [26].

From Table 1, we can make the following observations.

(1) DT models do not work well without learning embedding features, which is also observed in some other work [14]. This verifies the strength of semantic matching reported in the recent literature.

(2) We were unable to achieve good results using CNN after extensive tuning. This shows the limited advantage of CNN over standard text embeddings in the evaluated dataset. More importantly, the training time is significantly longer than standard neural architectures. MCN, on the other hand, adds minimal training overhead due to the simple operations involved.

(3) The positive results of Multiplication First shows that early matching, while simple, provides meaningful performance improvements over concatenation. This empirically verifies our findings in Section 3 that the concatenation architecture is inefficient in learning the query-document interactions.

(4) The insignificant results of Latent Cross shows that simply treating the query as a latent feature does not work well. Unlike in recommendation systems [4], we need to explicitly model query-document matching to attain the best results in search ranking.

(5) MCN and Kernel MCN work the best among all the models. The experiment numbers show that 1) early matching matters, and 2) facilitating interaction between the output of matching and side information is meaningful, as MCN outperforms the Multiplication First method on both the GMail and Drive datasets.

(6) The minor difference between kernel MCN and MCN shows that simply increasing the capacity of the matching function does not significantly increase performance. Early matching itself, though simple, works sufficiently well on the large-scale real-world datasets.

Table 1: Relative improvements compared to the baseline model. Bold numbers denote statistically significant differences versus the baseline (at the $p < 0.05$ level using a two-tailed t-test). † denote statistically significant improvement over Multiplication First. Note that DT does not include query and document embedding features (see text).

Model	Drive WMRR	GMail WMRR
DT	**-1.19%**	**-0.77**%
CNN	+0.07%	+0.01%
Latent Cross Query	-0.07%	-0.08%
Multiplication First	**+0.28**%	**+0.27**%
MCN	**+0.47**%†	**+0.31**%
Kernel MCN	**+0.49**%†	**+0.30**%

6.4 Online Experimental Result

We conducted large-scale online experiments to further verify the effectiveness of the MCN method. We use Drive search for the experiment, as MCN achieves the best improvement over the baseline on this dataset offline. The experiments were run in a standard A/B experiment setting for a subset of users for one month in March 2019, collecting metrics from 10 million search sessions. We allocated a fraction of online traffic to the experiments, half of the traffic uses existing production (Concatenation) model and half uses the MCN model. Because MCN has minimal changes compared to the production model, it easily met the latency requirements necessary to support Drive's "search-as-you-type" experience that serves millions of users. Note that the baseline is trained using the same training data as the experimental model. The results are shown in Table 2.

Table 2: Relative performance of MCN over the production model (Concatenation), together with the confidence interval (CI). Statistically significant results are bold. ↓ indicates that a lower value reflects better performance.

Metric	Relative Performance	CI
CTR	**+1.90%**	±1.42%
ACP (↓)	**-1.57%**	±0.64%
Abandon Rate (↓)	**-0.74%**	±0.69%
Long Click per Session	+0.96%	±1.54%
Time to Click (↓)	-0.62%	±1.64%

We see a consistent 1.9% relative CTR (click through rate) improvement and 1.57% ACP (average click position) decrease compared with the production baseline model. These improvements are considered highly significant in the production setting. It shows that MCN generalizes well beyond optimizing logged click data.

The other three metrics are session-level metrics. Abandon rate is the percentage of users abandoning a search session without clicking any results. Long Click per Session is the number of long clicks in every search session (dwelling on the clicked document for more than five seconds). Time to Click is the time a user spends between starting a search session and the first click. All these metrics are significantly positive or trending positive, showing MCN helps to improve the user's search experience. MCN has been fully deployed to serve all Drive search traffic due to these positive evaluation results and the ease of deployment.

7 CONCLUSIONS

We studied how to utilize query and document embeddings more effectively and efficiently in the presence of other side information in learning to rank problems. We started by showing the limitation of standard neural network architectures with synthetic data that imitates search scenario. Motivated by the findings, we designed the Matching Cross Network (MCN), a simple but effective model architecture that results in significant search quality benefits in two of the world's largest personal search engines.

REFERENCES

[1] Martín Abadi et al. 2016. TensorFlow: A System for Large-scale Machine Learning. In *USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 265–283.

[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 135–144.

[3] Michael Bendersky, Xuanhui Wang, Donald Metzler, and Marc Najork. 2017. Learning from user interactions in personal search via attribute parameterization. In *International Conference on Web Search and Data Mining (WSDM)*. 791–799.

[4] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *International Conference on Web Search and Data Mining (WSDM)*. 46–54.

[5] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search engines: Information retrieval in practice*. Vol. 283.

[6] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *International Conference on Web Search and Data Mining (WSDM)*.

[7] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W Bruce Croft. 2017. Neural ranking models with weak supervision. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 65–74.

[8] Pinar Donmez, Krysta M Svore, and Christopher JC Burges. 2009. On the local optimality of LambdaRank. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 460–467.

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

[10] Faezeh Ensan and Ebrahim Bagheri. 2017. Document Retrieval Model Through Semantic Linking. In *International Conference on Web Search and Data Mining (WSDM)*. 181–190.

[11] Mihajlo Grbovic. 2017. Search Ranking And Personalization at Airbnb. In *ACM Conference on Recommender Systems (RecSys)*. 339–340.

[12] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *Conference on Information and Knowledge Management (CIKM)*. 2333–2338.

[13] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[14] Pan Li, Zhen Qin, Xuanhui Wang, and Donald Metzler. 2019. Combining Decision Trees and Neural Networks for Learning-to-Rank in Personal Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 2032–2040.

[15] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[16] Bhaskar Mitra and Nick Craswell. 2018. An Introduction to Neural Information Retrieval. *Foundations and Trends in Information Retrieval* 13, 1 (2018), 1–126.

[17] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *The World Wide Web Conference (WWW)*. 1291–1299.

[18] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching As Image Recognition. In *AAAI Conference on Artificial Intelligence (AAAI)*. 2793–2799.

[19] Alberto Purpura, Marco Maggipinto, Gianmaria Silvello, and Gian Antonio Susto. 2019. Probabilistic Word Embeddings in Neural IR: A Promising Model That Does Not Work as Expected (For Now). In *International Conference on the Theory of Information Retrieval (ICTIR)*.

[20] Jinfeng Rao, Wei Yang, Yuhao Zhang, Ferhan Ture, and Jimmy Lin. 2019. Multi-perspective relevance matching with hierarchical convnets for social media search. In *AAAI Conference on Artificial Intelligence (AAAI)*. 232–240.

[21] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. 2004. Simple BM25 Extension to Multiple Weighted Fields. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 42–49.

[22] Jiaming Shen, Maryam Karimzadehgan, Michael Bendersky, Zhen Qin, and Donald Metzler. 2018. Multi-task learning for email search ranking with auxiliary query clustering. In *Conference on Information and Knowledge Management (CIKM)*. 2127–2135.

[23] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *The World Wide Web Conference (WWW)*. 373–374.

[24] BartTwardowski. 2016. Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks. In *ACM Conference on Recommender Systems (RecSys)*. 273–276.

[25] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 515–524.

[26] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to rank with selection bias in personal search. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 115–124.

[27] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270.

[28] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. 55–64.

[29] Dawei Yin et al. 2016. Ranking Relevance in Yahoo Search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 323–332.

[30] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational context for ranking in personal search. In *The World Wide Web Conference (WWW)*. 1531–1540.

[31] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. 2018. Neural Ranking Models with Multiple Document Fields. In *International Conference on Web Search and Data Mining (WSDM)*. 700–708.

[32] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to Do Next: Modeling User Behaviors by time-LSTM. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 3602–3608.