Separate and Attend in Personal Email Search

Yu Meng[1*], Maryam Karimzadehgan[2], Honglei Zhuang[2], Donald Metzler[2]
[1]Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
[2]Google LLC, Mountain View, CA, USA
[1]yumeng5@illinois.edu [2]{maryamk, hlz, metzler}@google.com

ABSTRACT

In personal email search, user queries often impose different requirements on different aspects of the retrieved emails. For example, the query "my recent flight to the US" requires emails to be ranked based on both textual contents and recency of the email documents, while other queries such as "medical history" do not impose any constraints on the recency of the email. Recent deep learning-to-rank models for personal email search often directly concatenate dense numerical features[1] (*e.g.*, document age) with embedded sparse features (*e.g.*, n-gram embeddings). In this paper, we first show with a set of experiments on synthetic datasets that direct concatenation of dense and sparse features does not lead to the optimal search performance of deep neural ranking models. To effectively incorporate both sparse and dense email features into personal email search ranking, we propose a novel neural model, **SepAttn**. **SepAttn** first builds two separate neural models to learn from sparse and dense features respectively, and then applies an attention mechanism at the prediction level to derive the final prediction from these two models. We conduct a comprehensive set of experiments on a large-scale email search dataset, and demonstrate that our **SepAttn** model consistently improves the search quality over the baseline models.

CCS CONCEPTS

• **Information systems** → **Learning to rank**; • **Computing methodologies** → *Neural networks*; *Regularization.*

KEYWORDS

Learning-to-Rank, Email Search, Neural Attention Model

ACM Reference Format:
Yu Meng[1*], Maryam Karimzadehgan[2], Honglei Zhuang[2], Donald Metzler[2]. 2020. Separate and Attend in Personal Email Search. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM'20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

[1]Throughout the paper, we use dense features to refer to dense numerical features that cannot be derived from sparse features; not to be confused with embedded sparse features.

*Work done while Yu Meng was at Google.

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

Email has long been an important means of daily communication. Personal email search, which helps users to quickly retrieve the emails they are looking for from their own corpora, has been an intriguing research topic in information retrieval (IR) for years. Email search is formulated as a learning-to-rank problem, which has been tackled with different learning models, such as boosted trees [8], SVM-based linear models [10, 12, 23], and shallow neural networks [9, 11].

Recently, deep neural networks (DNNs) have shown great success in learning-to-rank tasks. They significantly improve the performance of search engines in the presence of large-scale query logs in both web search [19] and email settings [39, 45, 51]. The advantages of DNNs over traditional models are mainly two-fold: (1) DNNs have strong power to learn embedded representations from sparse features, including words [32] and characters [6]. This allows effective and accurate matching of textual features between queries and documents. (2) DNNs are proved to have universal approximation capability [21], allowing to capture high-order interactions between query and document features.

In the personal email search scenario, user queries impose different requirements on different aspects of email documents to be retrieved. For example, the query "my recent flight to the US" requires the email search system to focus on both the textual contents and the recency of email documents, while queries such as "medical history" expect emails to be retrieved regardless of the recency. In email search models, different properties of email documents are reflected by different types of features, including dense numerical ones (*e.g.*, document age) and sparse categorical ones (*e.g.*, n-grams). However, there have been few efforts that study how to effectively combine dense features with sparse features in the learning-to-rank setting, probably because a natural approach exists—simply concatenating dense features with embedded sparse features and feeding them into the DNNs. Indeed, many previous deep neural email search models use direct concatenation of dense features with embedded sparse features [15, 38, 39, 45].

In this paper, we first begin with a set of empirical findings and analyses to show that direct concatenation of dense with embedded sparse features does not lead to the optimal performance of DNN-based ranking models. As a result, we propose the **SepAttn** model as an effective way to incorporate both dense and sparse features into the personal email search model. More specifically, **SepAttn** model consists of three major modules: **(1)** Separate DNN models to learn from sparse and dense features, respectively. **(2)** An attention mechanism that aggregates the outputs from the sparse feature and dense feature DNN models. **(3)** A regularizer that enables the joint learning of the sparse and dense feature DNN models. The main advantage of **SepAttn** over the simple concatenation approach is

that **SepAttn** separates dense from sparse features, and automatically learns to *explicitly* focus on the important feature set, while ignore the unimportant ones for different query types.

In summary, the followings are the contributions of this paper:

- We empirically show that simply concatenating embedded sparse features with dense numerical features is sub-optimal for DNN-based neural ranking models.
- To effectively leverage both sparse and dense features in neural ranking models, we propose to learn two separate models where their outputs are aggregated via an **attention** mechanism to derive the final output.
- We propose a regularization method to train the sparse feature model and dense feature model in a collaborative manner.
- We conduct a comprehensive set of experiments on synthetic datasets and a large-scale real-world email search dataset to demonstrate the advantage of our proposed method.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 motivates the problem through a set of experiments on synthetic datasets. Section 4 introduces our proposed **SepAttn** model. Section 5 presents and analyzes the performances of **SepAttn** on the synthetic datasets. In Section 6, we report and discuss the experimental results on a real-world large-scale email search dataset. Section 7 concludes the paper and discusses future directions.

2 RELATED WORK

In this section, we review related works on learning-to-rank, email search models and state-of-the-art neural attention models.

2.1 Learning-to-Rank

Learning-to-rank refers to building ranking models with machine learning algorithms. In early years, learning-to-rank has been studied with different models, such as boosted trees [8], SVM-based linear models [10, 12, 23] and shallow neural networks [9, 11]. Recent years have witnessed great success of applying DNNs to learning-to-rank, such as [7, 15, 35, 38]. For a complete literature review on neural ranking models for information retrieval, please refer to a survey by Mitra and Craswell [33].

2.2 Email Search

There have been several studies in the IR community focusing on the task of email search. The Enterprise tracks of TREC 2005 [40] and TREC 2006 [41] provide public datasets containing email data and summarize some early explorations [14, 31, 34]. A typical trade-off in email search system is to balance the importance of content-based relevance and other features, *e.g.* freshness. Carmel et al. [12] proposed an email search framework with a learning-to-rank re-ranking module combining freshness with relevance signals of emails as well as other features such as user actions. Alternatively, Carmel et al. [13] studied to present users with both the relevance-ranked results as well as the time-ranked results in two separate lists for better user experience. A number of studies specifically focus on improving the content-based relevance signals in email search. Kuzi et al. [27] explored several methods to expand the usually short and sparse queries by finding more related terms to improve the relevance results. Li et al. [29] studied a more specific synonym expansion problem to improve email search performance.

User interaction data such as clicks is another important signal for learning-to-rank models in email search. Bendersky et al. [4] leveraged user interactions by attribute parameterization. Wang et al. [48] mitigated the position bias in click data for better training of the model. In addition, Zamani et al. [51] showed that contexts such as search request time and location of users were helpful for email search quality.

There are also studies on understanding and leveraging query intent information in email search. Ai et al. [2] conducted a thorough survey of search intent by analyzing user logs of email search. Shen et al. [39] categorized email search queries into different clusters before adding the query cluster information to improve email ranking.

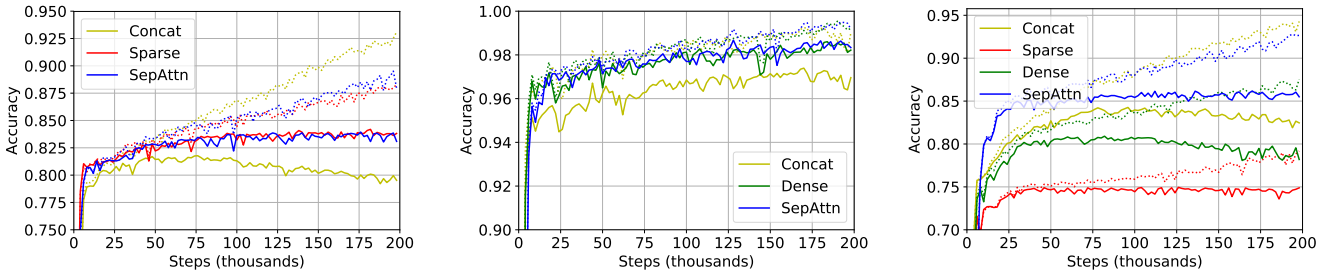
To the best of our knowledge, there is no previous work on email search that studied how to effectively combine dense numerical features with embedded sparse features in DNN-based ranking models. In the previous works, dense features were directly concatenated with embedded sparse features, which led to the suboptimal performances of DNN ranking models, as we will show later.

2.3 Attention Models

Recently, neural attention mechanisms have demonstrated enormous power on sequence modeling. They derived the optimal sequence representation by learning to focus on the important tokens in the sequence and down-weighting unimportant ones for downstream tasks. The attention mechanism was first proposed by Bahdanau et al. [3] in machine translation, where attention was used on top of RNN encoders for input-output alignment. Later, the attention mechanism has been adapted to a wide range of compelling sequence modeling tasks, including image caption generation [49], text classification [50] and natural language question answering [20, 26, 43].

The above studies employ attention mechanism in conjunction with RNN or CNN models. Vaswani et al. [46] proposed the Transformer, which used self-attention along with positional encoding. Later, Devlin et al. [16] proposed a deep bidirectional Transformer structure, BERT, which becomes one of the state-of-the-art pre-trained language models benefiting many downstream tasks with fine-tuning. The attention mechanism has also been generalized to attend to a group of structurally adjacent items instead of single ones, as studied by Li et al. [30].

Regardless of the manner that attention mechanisms are used in the previous works, the common purpose was to derive sequence representations as a weighted average of token representations. Therefore, the attention mechanisms were applied at the *feature level*, as a step of feature learning. In our work, however, the attention is used to aggregate the outputs from two models—the sparse and the dense feature models, to derive the final scoring outputs of the ranking model. To the best of our knowledge, this is the first work that applies attention mechanism at the *prediction level*.



(a) First Synthetic dataset (Sparse only)

(b) Second Synthetic dataset (Dense only)

(c) Third Synthetic dataset (Sparse and Dense Combined)

Figure 1: Training and testing performance of three models, *i.e.*, Sparse only, Dense only and Sparse + Dense models on three synthetic datasets. The dotted lines represent training accuracy and the solid lines represent testing accuracy. For completeness, we also show the performance of our SepAttn model in these figures. "Accuracy" is the ratio of correctly classified documents.

3 MOTIVATION

In this section, we provide motivation for our proposed "Separate and Attend" model, *i.e.*, **SepAttn**, and show that simply concatenating different feature types (*i.e.*, dense and sparse features) does not lead to the optimal performance. Different queries in email search impose different requirements. For example, a query such as "medical history" does not indicate requirements on dense features (*e.g.*, recency of an email document). As a result, simply concatenating both dense and sparse features and feeding them into the model might lead to suboptimal performance for such queries. In such a case, dense features are purely noise and could potentially distract the model from concentrating on useful features.

In order to simulate the personal email search scenario, we create three synthetic datasets and consider a simple binary classification problem, where a document d is classified to be relevant ($y = 1$) or irrelevant ($y = 0$) for a given query q. We use 100-dimensional GloVe pretrained embeddings [37] as embedded sparse feature representations for queries and documents.

The query features $\boldsymbol{q}$ and document sparse features $\boldsymbol{d}_{\text{sparse}}$ are obtained by randomly sampling word embeddings from the pretrained embedding vocabulary. The document dense features $\boldsymbol{d}_{\text{dense}}$ are generated from a uniform distribution in the interval $[-0.5, 0.5]$. We set the dimension of $\boldsymbol{d}_{\text{sparse}}$ (after embedding) and $\boldsymbol{d}_{\text{dense}}$ to be both 100. The ground-truth labels are generated differently for each of these synthetic datasets which will be explained later in this section.

We compare the performance of these models: **(1)** The model that learns from $\boldsymbol{q}$ and $\boldsymbol{d}_{\text{sparse}}$ only; **(2)** The model that learns from $\boldsymbol{q}$ and $\boldsymbol{d}_{\text{dense}}$ only; **(3)** The model that learns from $\boldsymbol{q}$ and the concatenation of $\boldsymbol{d}_{\text{sparse}}$ and $\boldsymbol{d}_{\text{dense}}$. All models are built based on a feed-forward DNN [28] with a hidden layer of 50 dimensions; the only difference between the models is that they learn from different feature sets. We generate 20,000 samples for both training and testing sets.

For the **first** dataset, the following rule describes how the ground-truth data is generated:

$$y = \begin{cases} 1, & \text{if } \cos(\boldsymbol{d}_{\text{sparse}}, \boldsymbol{q}) > 0 \\ 0, & \text{else} \end{cases}$$

which simulates the scenario where document and query matching is based only on sparse features, and dense features are unimportant/unused.

Similarly, we create the **second** synthetic dataset by generating ground-truth labels only based on dense features as below:

$$y = \begin{cases} 1, & \text{if } \boldsymbol{q} \in Q \text{ and } \sum \boldsymbol{d}_{\text{dense}} < 0 \\ 0, & \text{else} \end{cases}$$

where Q is a manually selected vocabulary of words that impose requirements on the recency of the documents, such as "recent", "latest", "newest", "today".

We show the performance of the aforementioned models on the first synthetic dataset and second synthetic dataset in Figure 1(a) and Figure 1(b), respectively. In these figures, dotted lines represent training curves, and solid lines represent testing curves. We also show the performance of our proposed method **SepAttn** for completeness in these figures and postpone the discussions about its performance to Section 5.

On the first synthetic dataset, sparse only model is the benchmark because it learns only from sparse features and inherently avoids unimportant dense features. We observe from Figure 1(a) that the concatenation model overfits the training set—when its training accuracy goes up, its testing accuracy goes down. This indicates that the concatenation model learns noisy signals from the training set that cannot be generalized to the testing set, demonstrating its ineffectiveness of discriminating the useful features from unimportant ones.

On the second synthetic dataset, dense only model is the benchmark since it avoids the unimportant sparse features. Figure 1(b) shows that the testing accuracy of concatenation model falls behind that of the dense only model. This again demonstrates that the concatenation model is negatively influenced by unimportant features.

The **third** synthetic dataset is generated based on combination of both sparse and dense features and the ground-truth labels are defined as follows:

$$y = \begin{cases} 1, & \text{if } \big(\cos(\boldsymbol{d}_{\text{sparse}}, \boldsymbol{q}) > 0\big) \wedge \big(\boldsymbol{q} \in Q \text{ and } \sum \boldsymbol{d}_{\text{dense}} < 0\big) \\ 0, & \text{else} \end{cases}$$

where the document-query matching requires both sparse and dense feature matching.

The performance of the above models using this dataset is shown in Figure 1(c). Although the concatenation model outperforms sparse only and dense only models, it still suffers from overfitting (testing accuracy increases while training accuracy decreases). This demonstrates that DNNs are ineffective to learn from concatenated sparse and dense features. We explain the reason as below: $\boldsymbol{d}_{\text{dense}}$ and $\boldsymbol{d}_{\text{sparse}}$ come from different feature space—dense numerical feature values (*e.g.*, document age) have practical meanings (*i.e.*, a higher value of document age means the email was received earlier), while embedding feature values (*e.g.*, n-gram embedding) do not have practical meanings [42] (*i.e.*, a higher value in n-gram embedding dimensions is not interpretable). If $\boldsymbol{d}_{\text{dense}}$ and $\boldsymbol{d}_{\text{sparse}}$ are directly concatenated and fed into the DNNs, operations applied between $\boldsymbol{d}_{\text{dense}}$ and $\boldsymbol{d}_{\text{sparse}}$ are meaningless (*e.g.*, it does not make sense to add/multiply document age with n-gram embeddings) and can lead to ineffectiveness and overfitting of DNNs.

In the next section, we describe the details of our model, *i.e.*, **SepAttn**.

4 METHODOLOGY

We first formulate the problem and define the notations. Then we describe our **SepAttn** model.

4.1 Problem Formulation

The inputs for personal email search problem are tuples $(q, \mathcal{D})$ where q is a user query string, and $\mathcal{D} = \{d_1, \ldots, d_n\}$ is a list of candidate documents. The ground truth labels $\boldsymbol{y} \in \{0, 1\}^n$ are user click-through data indicating whether the corresponding document is clicked. The goal of personal email search model is to rank $\mathcal{D}$ so that the clicked document is ranked as high as possible.

The query features $\boldsymbol{q}$ are n-grams and character n-grams extracted from the original user query string q, the document features $\boldsymbol{d}$ include sparse categorical features $\boldsymbol{d}_{\text{sparse}}$ (*e.g.*, n-grams) and dense numerical features $\boldsymbol{d}_{\text{dense}}$ (*e.g.*, document ages).

To preserve privacy, the query-document inputs are anonymized based on k-anonymity approach [44], and only query and document n-grams that are frequent in the entire corpus are retained. We summarize the features used in the model in Table 1.

Given the input features $\mathcal{X} = \{\boldsymbol{q}, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_n\}$, the email search model learns a per-item scoring function $h : (\boldsymbol{q}, \boldsymbol{d}_i) \to \mathbb{R}$, such that

$$\boldsymbol{h}(\mathcal{X}) = \big[h(\boldsymbol{q}, \boldsymbol{d}_1) \quad \cdots \quad h(\boldsymbol{q}, \boldsymbol{d}_n)\big]^\top \tag{1}$$

induces a permutation π and $\boldsymbol{h}(\mathcal{X})\big|_{\pi^{-1}(r)}$ monotonically decreases with increasing rank r. The per-item scoring function h could be learned by gradient boosted trees [18], support vector machines [24] or neural networks [9]. In this paper, we aim to design a DNN-based model that effectively incorporates both $\boldsymbol{d}_{\text{sparse}}$ and $\boldsymbol{d}_{\text{dense}}$ for improving personal email search performance.

With the input features $(\boldsymbol{q}, \boldsymbol{d}_i)$ described above, ground truth click-through data y_i and a per-item scoring function h, we follow the Softmax Cross-Entropy listwise loss setting in [36] to train the model. Specifically, the loss is defined as

$$\mathcal{L}_{\text{listwise}} = -\sum_{i=1}^{n} y_i \log\left(\frac{\exp\big(h(\boldsymbol{q}, \boldsymbol{d}_i)\big)}{\sum_{j=1}^{n} \exp\big(h(\boldsymbol{q}, \boldsymbol{d}_j)\big)}\right). \tag{2}$$

Table 1: Features used in personal email search.

Type	Features
Query Feature ($\boldsymbol{q}$)	n-grams character n-grams
Document Sparse Feature ($\boldsymbol{d}_{\text{sparse}}$)	n-grams character n-grams
Document Dense Feature ($\boldsymbol{d}_{\text{dense}}$)	number of attachments number of recipients document age

4.2 Separate and Attend Method

In this section, we introduce our "Separate and Attend model", *i.e.*, **SepAttn** model, to address the observed DNN learning issues when sparse and dense features are directly concatenated, as described in Section 3. The model structure of **SepAttn** is presented in Figure 2.

Often times different queries impose different requirements regarding an email document to be retrieved, and different email document properties are reflected by the sparse and dense features extracted. In this section, we first build separate models that learn from each sparse and dense features individually, and then develop an **attention** mechanism at the prediction stage that enables the model to explicitly learn to focus on the important features for different queries.

Specifically, we first build two models to learn two per-item scoring functions (one model for sparse and one model for dense features):

$$h_{\text{sparse}} : (\boldsymbol{q}, \boldsymbol{d}_{i:\text{sparse}}) \to \mathbb{R}, \quad h_{\text{dense}} : (\boldsymbol{q}, \boldsymbol{d}_{i:\text{dense}}) \to \mathbb{R},$$

which score a document given a query based on sparse and dense document features, respectively.

Then the scores of a list of n documents given by sparse and dense feature models are:

$$\boldsymbol{h}_{\text{sparse}}(\mathcal{X}) = \big[h_{\text{sparse}}(\boldsymbol{q}, \boldsymbol{d}_{1:\text{sparse}}) \quad \cdots \quad h_{\text{sparse}}(\boldsymbol{q}, \boldsymbol{d}_{n:\text{sparse}})\big]^\top,$$

$$\boldsymbol{h}_{\text{dense}}(\mathcal{X}) = \big[h_{\text{dense}}(\boldsymbol{q}, \boldsymbol{d}_{1:\text{dense}}) \quad \cdots \quad h_{\text{dense}}(\boldsymbol{q}, \boldsymbol{d}_{n:\text{dense}})\big]^\top.$$

Next, we introduce an **attention** mechanism that aggregates $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$ to derive the final scores for the document list. Specifically, we first feed $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$ to a one-layer feed-forward neural network [28] to derive the hidden representation of the document scores, and then compute their attention weights based on the hidden representations as well as a context vector. Finally, $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$ are weighted-averaged according to their attention weights to derive the final scores $\boldsymbol{h}(\mathcal{X})$ for all the documents. Mathematically, the **attention**

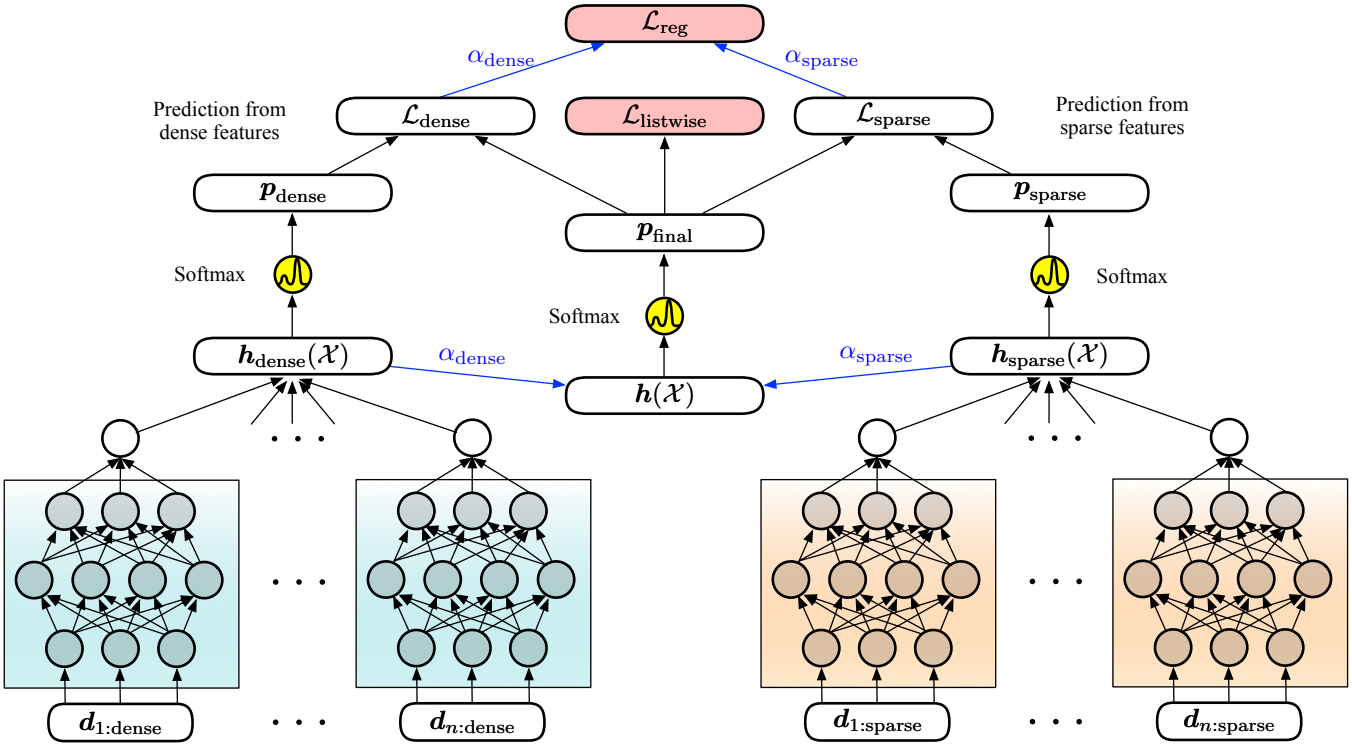


Figure 2: The overview of SepAttn model. SepAttn builds separate models to learn from sparse and dense features, and then aggregates the two models' outputs via an attention mechanism. SepAttn also employs a regularization term to enable joint training of the two models, and the final loss function is a combination of the listwise loss with the regularization loss. For simplicity, we omit the query features which are concatenated with the document dense/sparse features before fed into the DNN models.

mechanism is presented as:

$$\boldsymbol{u}_{\text{sparse}} = \tanh(W\boldsymbol{h}_{\text{sparse}}(\mathcal{X}) + \boldsymbol{b}),$$

$$\boldsymbol{u}_{\text{dense}} = \tanh(W\boldsymbol{h}_{\text{dense}}(\mathcal{X}) + \boldsymbol{b}),$$

$$\alpha_{\text{sparse}} = \frac{\exp(\boldsymbol{u}_{\text{sparse}}^{\top}\boldsymbol{v})}{\exp(\boldsymbol{u}_{\text{sparse}}^{\top}\boldsymbol{v}) + \exp(\boldsymbol{u}_{\text{dense}}^{\top}\boldsymbol{v})}, \tag{3}$$

$$\alpha_{\text{dense}} = \frac{\exp(\boldsymbol{u}_{\text{dense}}^{\top}\boldsymbol{v})}{\exp(\boldsymbol{u}_{\text{sparse}}^{\top}\boldsymbol{v}) + \exp(\boldsymbol{u}_{\text{dense}}^{\top}\boldsymbol{v})}, \tag{4}$$

$$\boldsymbol{h}(\mathcal{X}) = \alpha_{\text{sparse}}\boldsymbol{h}_{\text{sparse}}(\mathcal{X}) + \alpha_{\text{dense}}\boldsymbol{h}_{\text{dense}}(\mathcal{X}), \tag{5}$$

where $W \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$ are trainable weights and bias vector respectively; $\boldsymbol{v} \in \mathbb{R}^n$ is the context vector, randomly initialized and trained together with other parameters.

Here α_{sparse} and α_{dense} are **attention weights** assigned to the sparse feature model and the dense feature model, respectively. As shown in Equation (5), the higher the attention weight a model gets, the more dominant role it plays in the final result.

We explain why the above attention mechanism enables the model to focus on important features for different queries. The hidden representations $\boldsymbol{u}_{\text{sparse}}$ and $\boldsymbol{u}_{\text{dense}}$ can be interpreted as encoding the importance of $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$ respectively. For example, if dense features are noisy features (*i.e.*, not important for a given query which might mislead the model) with respect to a specific query, $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$ will be random across all documents, and $\boldsymbol{u}_{\text{dense}}$ will encode that this is an unimportant result. The context vector $\boldsymbol{v}$ can be seen as a representation of important results, and therefore a hidden representation similar to $\boldsymbol{v}$ will be assigned higher weights. In this way, the unimportant/noisy features will be down-weighted by the attention mechanism, and predictions from the important features will dominate the final score.

In the following section, we describe the model learning with regularization.

4.3 Joint Learning with Regularization

Until now, the dense and sparse feature models are trained without any interactions, *i.e.*, the outputs of dense feature model will not directly influence the training of the sparse feature model, and vice versa. To enable direct interactions between the two models, we introduce a regularization term into **SepAttn**, motivated by the idea of co-training [5] that when two learning models capture different and complementary feature sets of the same instances, the two models can mutually enhance each other during training.

To encourage the dense and sparse feature models to output consistent results given the same set of documents, a straightforward way is to minimize the difference between $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$. However, since the two sets of scores are computed from different feature sets, they will be in different scales, making it ineffective

to regularize directly on $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$. Therefore, we instead regularize on the probability distributions after normalizing $\boldsymbol{h}_{\text{sparse}}(\mathcal{X})$ and $\boldsymbol{h}_{\text{dense}}(\mathcal{X})$ with Softmax, *i.e.*,

$$\boldsymbol{p}_{\text{sparse}} = \left[\frac{\exp\left(h_{\text{sparse}}(\boldsymbol{q}, \boldsymbol{d}_{1:\text{sparse}})\right)}{\sum_{j=1}^{n} \exp\left(h_{\text{sparse}}(\boldsymbol{q}, \boldsymbol{d}_{j:\text{sparse}})\right)} \quad \cdots \quad \frac{\exp\left(h_{\text{sparse}}(\boldsymbol{q}, \boldsymbol{d}_{n:\text{sparse}})\right)}{\sum_{j=1}^{n} \exp\left(h_{\text{sparse}}(\boldsymbol{q}, \boldsymbol{d}_{j:\text{sparse}})\right)} \right]^{\top},$$

$$\boldsymbol{p}_{\text{dense}} = \left[\frac{\exp\left(h_{\text{dense}}(\boldsymbol{q}, \boldsymbol{d}_{1:\text{dense}})\right)}{\sum_{j=1}^{n} \exp\left(h_{\text{dense}}(\boldsymbol{q}, \boldsymbol{d}_{j:\text{dense}})\right)} \quad \cdots \quad \frac{\exp\left(h_{\text{dense}}(\boldsymbol{q}, \boldsymbol{d}_{n:\text{dense}})\right)}{\sum_{j=1}^{n} \exp\left(h_{\text{dense}}(\boldsymbol{q}, \boldsymbol{d}_{j:\text{dense}})\right)} \right]^{\top}.$$

We then minimize the KL divergence of $\boldsymbol{p}_{\text{sparse}}$ and $\boldsymbol{p}_{\text{dense}}$ with respect to the final prediction $\boldsymbol{p}_{\text{final}}$:

$$\mathcal{L}_{\text{sparse}} = D_{\text{KL}}(\boldsymbol{p}_{\text{final}} \parallel \boldsymbol{p}_{\text{sparse}}) = \sum_{i=1}^{n} p_{i:\text{final}} \log\left(\frac{p_{i:\text{final}}}{p_{i:\text{sparse}}}\right),$$

$$\mathcal{L}_{\text{dense}} = D_{\text{KL}}(\boldsymbol{p}_{\text{final}} \parallel \boldsymbol{p}_{\text{dense}}) = \sum_{i=1}^{n} p_{i:\text{final}} \log\left(\frac{p_{i:\text{final}}}{p_{i:\text{dense}}}\right),$$

where $\boldsymbol{p}_{\text{final}}$ is **SepAttn**'s final prediction after normalizing $\boldsymbol{h}(\mathcal{X})$ in Equation (5) via Softmax.

The regularization loss becomes the weighted average of $\mathcal{L}_{\text{sparse}}$ and $\mathcal{L}_{\text{dense}}$:

$$\mathcal{L}_{\text{reg}} = \alpha_{\text{sparse}} \mathcal{L}_{\text{sparse}} + \alpha_{\text{dense}} \mathcal{L}_{\text{dense}}, \tag{6}$$

where α_{sparse} and α_{dense} directly come from Equations (3) and (4), respectively, and are automatically learned during training.

The purpose of including the attention weights α_{sparse} and α_{dense} again in the regularization term is to prevent noisy/unimportant features from misleading the regularization and disturbing the predictions learned from important features. For example, when dense features are noisy features with respect to a specific query, $\boldsymbol{p}_{\text{dense}}$ is a noisy distribution which is not meaningful to be regularized. Since the attention mechanism automatically down-weights unimportant features, the regularization term will encourage the interaction between the two models only when both feature sets are important.

Finally, the **SepAttn** model is trained via a combination of the listwise loss (Equation (2)) with the regularization loss:

$$\mathcal{L} = \mathcal{L}_{\text{listwise}} + \lambda \mathcal{L}_{\text{reg}}, \tag{7}$$

where $\lambda > 0$ is a hyperparameter that controls the importance of regularization. We will study its effect in model training in the experiment section.

5 RESULTS ON SYNTHETIC DATASETS

In this section, we describe the results of our method on synthetic datasets described in Section 3. From Figures 1(a) and 1(b), we observe that **SepAttn** achieves comparable performance on the testing set to Sparse-only (benchmark on the first synthetic dataset) and Dense-only (benchmark on the second synthetic dataset) models, respectively, which shows the robustness of our model against noisy/unimportant features. To understand how **SepAttn** is able to focus on important features and ignore noisy ones, we visualize the attention weights, *i.e.*, α_{sparse} and α_{dense} in Equations (3) and (4) on the first and second synthetic datasets. We randomly select 20 samples. It can be observed from Figure 3(a) that on the first synthetic dataset, **SepAttn** assigns almost zero attention weights to the dense features, and effectively focuses on the sparse features. This explains why **SepAttn** can achieve similar performance with the Sparse-only model. Similarly, as shown in Figure 3(b), **SepAttn** focuses mainly on the dense features for the second synthetic dataset, and does not pay attention to the sparse features.

In Figure 1(c), **SepAttn** performs the best on the testing set, demonstrating its effectiveness on learning from the combination of sparse and dense features. We visualize the attention weights in Figure 3(c), where we use 0/1 values to denote the classification outputs from sparse and dense feature models. For example, the first sample has a 0 in the sparse row, and a 1 in the dense row. This means that for the first sample, the sparse feature model predicts $y = 0$ and the dense feature model predicts $y = 1$. Recall that the third synthetic dataset is generated from both sparse and dense features, *i.e.*, only when both dense and sparse document features match with the query features, the ground-truth label is 1. In this case, the **SepAttn** model mainly focuses on the 0 predictions as shown in Figure 3(c), which explains why **SepAttn** works as expected—the final prediction is dominated by negative predictions from either dense or sparse features.

6 EXPERIMENTS

In this section, we begin with a description of the datasets we use in our experiments. Then, we evaluate our proposed technique and baselines using the evaluation metrics that we will define. Finally, we discuss the hyperparameter sensitivity of our method.

6.1 Dataset

Due to the private and sensitive nature of personal email data, there is no publicly available large-scale email search dataset. Therefore, the data we use comes from the search click logs of Gmail search engine. The training set contains around 317 million queries, and the testing set contains around 41 million queries. All queries in the testing set are issued strictly later than all queries in the training set. Each query has six candidate documents (*i.e.*, $n = 6$ in Equation (1)), one of which is clicked[2]. The goal is to rank the six documents to increase the likelihood of a higher ranked document being clicked.

6.2 Model Evaluation

In this subsection, we describe the evaluation metrics. We denote the evaluation set as Q, and the ranking position of the clicked document for query q as r_q^*.

- Mean reciprocal rank (MRR) of the clicked document:

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q^*}.$$

- Weighted mean reciprocal rank (WMRR) [47] of the clicked document:

$$\text{WMRR} = \frac{1}{\sum_{q \in |Q|} w_q} \sum_{q \in Q} \frac{w_q}{r_q^*},$$

where w_q is the bias correction weight, and is inversely proportional to the probability of observing a click at the clicked

[2]These six candidate documents are presented in the dropdown menu of the Gmail search box while users type their queries but before they click the search bottom. When users find the target email, the system will direct them to the exact email, generating exactly one click.

Table 2: Evaluations on testing set with percentage of improvements over the best baseline. Metrics with an upper arrow (↑) indicate higher is better; metrics with a down arrow (↓) indicate lower is better. Improvements with an asterisk (*) denotes statistical significance relative to the best performing baseline (Concatenation), according to the two-tailed paired t-test.

Methods	MRR (↑)	WMRR (↑)	ARP (↓)	WARP (↓)	DCG (↑)
Dense-only method	0.650	0.563	2.143	2.474	0.759
Sparse-only method	0.665	0.578	2.067	2.412	0.767
Concatenation method	0.682	0.589	2.002	2.366	0.770
SepAttn	**0.686**	**0.595**	**1.992**	**2.350**	**0.781**
$\Delta(\%)$	$+0.59^*$	$+1.02^*$	-0.50^*	-0.68^*	$+1.43^*$

position of q. We set those weights using result randomization, as described in [47]. This serves as our main evaluation metric.

- Average relevance value position (ARP) [52]:

$$\text{ARP} = \frac{1}{|Q|} \sum_{q \in Q} r_q^*.$$

- Weighted average relevance value position (WARP), which is the weighted version of the above ARP metric:

$$\text{WARP} = \frac{1}{\sum_{q \in |Q|} w_q} \sum_{q \in Q} w_q r_q^*,$$

where w_q is the bias correction weight as in WMRR.

- Discounted Cumulative Gain (DCG) [22]:

$$\text{DCG} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\log_2(1 + r_q^*)}.$$

6.3 Results and Discussions

In this section, we compare our proposed approach, *i.e.*, **SepAttn** model with the following baseline models.

- **Dense-only method**: The DNN model that only uses dense document features.
- **Sparse-only method**: The DNN model that only uses sparse document features.
- **Concatenation method**: The DNN model that learns from concatenated dense and embedded sparse features.

The above models are implemented using TF-ranking [36] which is a scalable open-source learning-to-rank Tensorflow library [1]. Due to the data anonymization process (*i.e.*, bag of frequent n-grams), the sequential information in the raw email documents is lost, and thus sequence-aware models like CNNs [38] and RNNs [35] cannot be applied to our case. The configuration of the DNN model is described below: The DNN model has three hidden layers whose dimensions are 256, 128, and 64, respectively. The n-grams and character n-grams are first passed through an embedding layer of size 20 followed by an average pooling layer before fed into the DNN. We use Adagrad [17] with 0.1 learning rate and batch size 100 to train the model. The above hyperparameters are the optimal settings of the **Concatenation method**, obtained from tuning the model in the following hyperparameter ranges: Layer dimension in $\{64, 128, 256, 512, 768, 1024\}$; embedding dimension in $\{20, 30, 40, 50, 100\}$; learning rate in $\{0.05, 0.08, 0.1, 0.15, 0.3\}$; optimization algorithm in {Adam [25], Adagrad [17]}.

We set the regularization weight to be 1 in our **SepAttn** model, and will study model sensitivity to this hyperparameter in Section 6.4.

The results are presented in Table 2. From the table, we can see that across all metrics, the concatenation of dense features with embedded sparse features (*i.e.*, Concatenation method) consistently leads to better results than both Sparse-only and Dense-only models. An interesting finding is that even without any sparse features (Dense-only), the model achieves reasonably good performance, probably because emails that are more recent or with more attachments are more likely to be the user-desired ones.

Moreover, our **SepAttn** method achieves statistically significant improvements (using the two-tailed paired t-test with 99% confidence level) over the Concatenation method.

We note that an improvement of 1% is considered to be highly significant for our email search ranking system. In Table 2, we see that the improvement of WMRR metric for **SepAttn** method is 1.02% over the best performing baseline model (Concatenation method).

The results demonstrate that dense features indeed play a critical role in email search tasks, and they need to be incorporated into the model effectively, *i.e.*, simply concatenating them with embedded sparse features is suboptimal; **SepAttn** provides a more effective way of learning from both sparse and dense features.

6.4 Sensitivity Analysis

In this section, we discuss the sensitivity of **SepAttn** method to the regularization parameter λ in Equation (7). We vary the regularization parameter λ in the range $[0, 2.5]$ and report **SepAttn**'s performance (with respect to WMRR metric) in Figure 4.

We observe that when $\lambda > 0$, the performance is significantly better than that when $\lambda = 0$. This shows that jointly training the sparse and dense feature models by encouraging them to make consistent prediction is indeed beneficial for improving the ranking performance. When λ continues to grow larger, the performance of the model becomes stable, which is a favorable property because λ can be safely set within a relatively wide range of values.

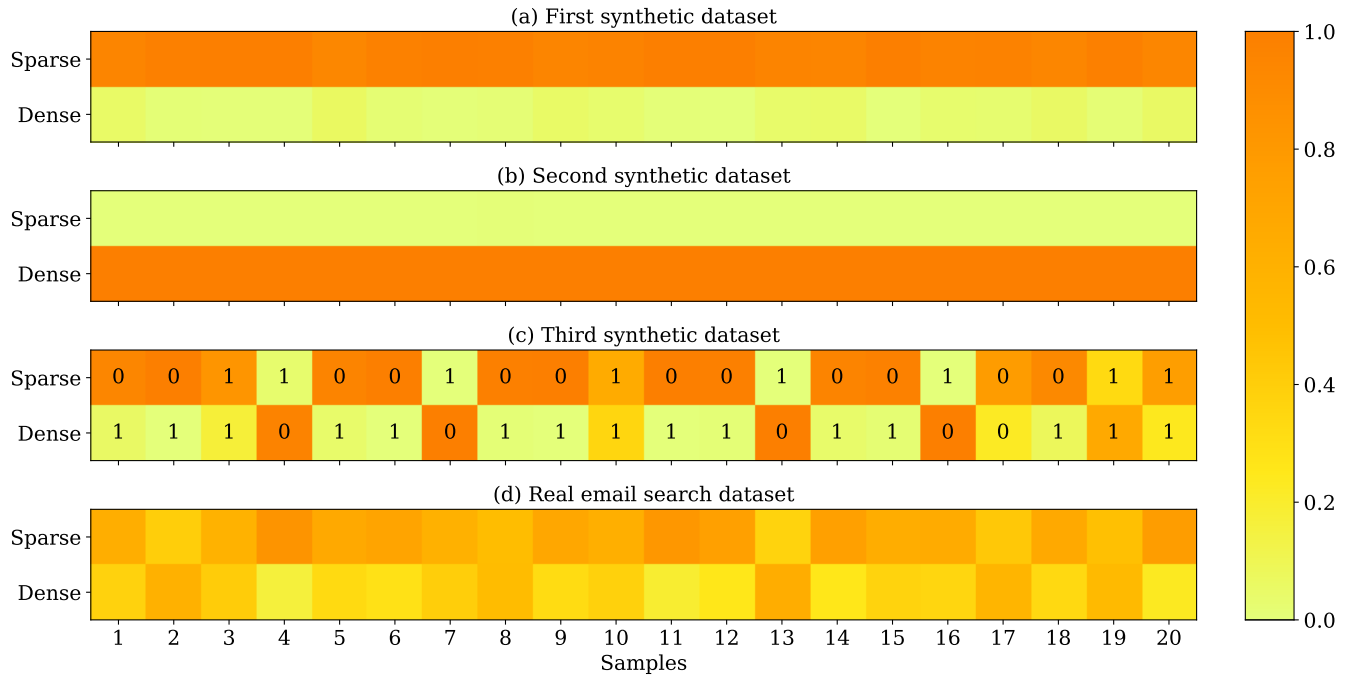


Figure 3: Attention weights visualization (α_{sparse} and α_{dense} from Equations (3) and (4)) over 20 randomly selected samples on four datasets: (a) First synthetic dataset; (b) Second synthetic dataset; (c) Third synthetic dataset; (d) Real email search dataset. The 0/1 values on (c) indicate classification outputs from sparse and dense feature models.

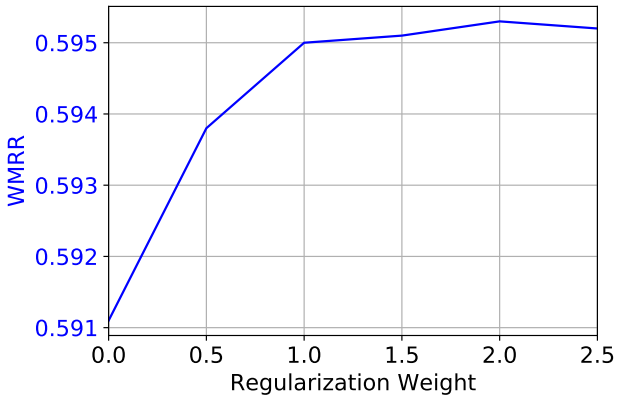


Figure 4: Sensitivity of regularization parameter λ on WMRR metric.

6.5 Attention Weights Visualization

To understand how **SepAttn** works for different email search queries, we visualize the attention weights (α_{sparse} and α_{dense} in Equations (3) and (4)) of **SepAttn** model on our email search dataset in Figure 3(d). We randomly select 20 queries. We observe the following: **(1)** Both sparse and dense features are important for email search tasks, and sparse features generally obtain higher weights than dense features, which corresponds to our intuition that textual features are more important than numerical features in email search. **(2)** Different queries have different attention weight distribution on sparse and dense features, which verifies the necessity to build an attentive ranking model that learns to focus on different feature sets according to different queries types. For example, the 2nd, 13th, 17th and 19th samples in Figure 3(d) have higher attention weights on dense features, while for other queries, sparse features are more important. **SepAttn** automatically learns to assign appropriate weights to dense and sparse features in order to achieve the best performance.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we studied how to improve email search ranking by effectively learning from a combination of dense and sparse document features using DNNs. We first showed on a set of synthetic datasets that simply concatenating dense features with embedded sparse features leads to suboptimal performance of DNN ranking models, mainly because these features are from different feature space. Motivated by this drawback, we proposed the **SepAttn** model which automatically learns to focus on important features for different queries through the **attention** mechanism. We evaluated our **SepAttn** model on a large-scale email search dataset and showed that it significantly outperforms the baseline approach—direct concatenation of dense features with sparse features.

In the future, we would like to extend our study to other IR tasks such as web search and recommendation. Our proposed method can be easily generalized to other scenarios where dense numerical

features and sparse categorical features both play a role. Furthermore, the attention mechanism at the prediction level introduced in this paper may facilitate further research on ensembling multiple learning models.

ACKNOWLEDGMENTS

We thank Zhongliang Li for his help on the model implementation. We thank anonymous reviewers for valuable and insightful feedback.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, and Jeffrey Dean. 2016. Tensorflow: a system for large-scale machine learning. *Operating Systems Design and Implementation* 16 (2016), 265–283.
[2] Qingyao Ai, Susan T Dumais, Nick Craswell, and Dan Liebling. 2017. Characterizing email search using large-scale behavioral logs and surveys. In *WWW*.
[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.
[4] Michael Bendersky, Xuanhui Wang, Donald Metzler, and Marc Najork. 2017. Learning from user interactions in personal search via attribute parameterization. In *WSDM*.
[5] Avrim Blum and Tom M. Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *COLT*.
[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2016), 135–146.
[7] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *WWW*.
[8] Christopher J. C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview.
[9] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent. In *ICML*.
[10] Yunbo Cao, Jun Xu, T. M. Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting ranking SVM to document retrieval. In *SIGIR*.
[11] Zhe Cao, Tao Qin, T. M. Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML*.
[12] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. 2015. Rank by Time or Relevance?: Revisiting Email Search. In *CIKM*.
[13] David Carmel, Liane Lewin-Eytan, Alex Libov, Yoelle Maarek, and Ariel Raviv. 2017. Promoting relevant results in time-ranked mail search. In *WWW*.
[14] Nick Craswell, Hugo Zaragoza, and Stephen Robertson. 2005. Microsoft cambridge at TREC-14: Enterprise track. In *TREC*.
[15] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and William Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *SIGIR*.
[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *HLT-NAACL*.
[17] John C. Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (2010), 2121–2159.
[18] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of statistics* (2001).
[19] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep Relevance Matching Model For Ad-hoc Retrieval. In *CIKM*.
[20] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *NIPS*.
[21] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4 (1991), 251–257.
[22] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20 (2002), 422–446.
[23] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *KDD*.
[24] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *KDD*.
[25] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[26] Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. In *ICML*.
[27] Saar Kuzi, David Carmel, Alex Libov, and Ariel Raviv. 2017. Query expansion for email search. In *SIGIR*.
[28] Yann LeCum, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521 (2015), 436–444.
[29] Cheng Li, Mingyang Zhang, Michael Bendersky, Hongbo Deng, Donald Metzler, and Marc Najork. 2019. Multi-view Embedding-based Synonyms for Email Search. In *SIGIR*.
[30] Yang Li, Lukasz Kaiser, Samy Bengio, and Si Si. 2019. Area attention. In *ICML*.
[31] Craig Macdonald and Iadh Ounis. 2006. Combining fields in known-item email search. In *SIGIR*.
[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*.
[33] Bhaskar Mitra and Nick Craswell. 2017. Neural Models for Information Retrieval. *ArXiv* abs/1705.01509 (2017).
[34] Paul Ogilvie and Jamie Callan. 2005. Experiments with Language Models for Known-Item Finding of E-mail Messages. In *TREC*.
[35] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. 2017. DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval. In *CIKM*.
[36] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Carlos Pignataro, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. In *KDD*.
[37] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*.
[38] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks. In *SIGIR*.
[39] Jiaming Shen, Maryam Karimzadehgan, Michael Bendersky, Zhen Qin, and Donald Metzler. 2018. Multi-Task Learning for Email Search Ranking With Auxiliary Query Clustering. In *CIKM*.
[40] Ian Soboroff, Arjen P de Vries, and Nick Craswell. 2005. Overview of the TREC 2005 Enterprise Track. In *TREC*.
[41] Ian Soboroff, Arjen P de Vries, and Nick Craswell. 2006. Overview of the TREC 2006 Enterprise Track. In *TREC*.
[42] Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard H. Hovy. 2017. SPINE: SParse Interpretable Neural Embeddings. In *AAAI*.
[43] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-To-End Memory Networks. In *NIPS*.
[44] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10 (2002), 557–570.
[45] Brandon Tran, Maryam Karimzadehgan, Rama Kumar Pasumarthi, Michael Bendersky, and Donald Metzler. 2019. Domain Adaptation for Enterprise Email Search. In *SIGIR*.
[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Lawrence Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS*.
[47] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *SIGIR*.
[48] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position bias estimation for unbiased learning to rank in personal search. In *WSDM*.
[49] Kelvin Xu, Jimmy Ba, Jamie Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *ICML*.
[50] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *HLT-NAACL*.
[51] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational Context for Ranking in Personal Search. In *WWW*.
[52] Mu Zhu. 2004. Recall, Precision and Average Precision.