Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks

Qingyao Ai
CICS, UMass Amherst
Amherst, MA, USA
aiqy@cs.umass.edu

Xuanhui Wang
Google Research
Mountain View, CA, USA
xuanhui@google.com

Sebastian Bruch
Google Research
Mountain View, CA, USA
bruch@google.com

Nadav Golbandi
Google Research
Mountain View, CA, USA
nadavg@google.com

Michael Bendersky
Google Research
Mountain View, CA, USA
bemike@google.com

Marc Najork
Google Research
Mountain View, CA, USA
najork@google.com

ABSTRACT

While in a classification or a regression setting a label or a value is assigned to each individual document, in a ranking setting we determine the relevance ordering of the entire input document list. This difference leads to the notion of relative relevance between documents in ranking. The majority of the existing learning-to-rank algorithms model such relativity at the loss level using pairwise or listwise loss functions. However, they are restricted to *univariate scoring functions*, i.e., the relevance score of a document is computed based on the document itself, regardless of other documents in the list. To overcome this limitation, we propose a new framework for *multivariate scoring functions*, in which the relevance score of a document is determined jointly by multiple documents in the list. We refer to this framework as GSFs—groupwise scoring functions. We learn GSFs with a deep neural network architecture, and demonstrate that several representative learning-to-rank algorithms can be modeled as special cases in our framework. We conduct evaluation using click logs from one of the largest commercial email search engines, as well as a public benchmark dataset. In both cases, GSFs lead to significant performance improvements, especially in the presence of sparse textual features.

CCS CONCEPTS

• **Information systems** → **Learning to rank**;

KEYWORDS

Multivariate scoring; groupwise scoring functions; deep neural architectures for IR

ACM Reference Format:
Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2019. Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks. In *The 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR '19), October 2–5, 2019, Santa Clara, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341981.3344218

ICTIR '19, October 2–5, 2019, Santa Clara, CA, USA

ACM ISBN 978-1-4503-6881-0/19/10.
https://doi.org/10.1145/3341981.3344218

1 INTRODUCTION

Unlike in classification or regression, the main goal of a ranking problem is not to assign a label or a value to individual items, but, given a list of items, to produce an ordering of the items in that list in such a way that the utility of the entire list is maximized. In other words, in ranking we are more concerned with the relative ordering of the relevance of items—for some notion of relevance—than their absolute magnitudes.

Modeling relativity in ranking has been extensively studied in the past, especially in the context of learning-to-rank [24]. Learning-to-rank aims to learn a scoring function that maps feature vectors to real-valued scores in a supervised setting. Scores computed by such a function induce an ordering of items in the list. The majority of existing learning-to-rank algorithms learn a parameterized function by optimizing a loss that acts on pairs of items (*pairwise*) or a list of items (*listwise*) [5, 7, 8, 37]. The idea is that such loss functions guide the learning algorithm to optimize preferences between pairs of items or to maximize a ranking metric such as NDCG [6, 20, 32], thereby indirectly modeling relative relevance.

Though effective, most existing learning-to-rank frameworks are restricted to the paradigm of *univariate* scoring functions: the relevance of an item is computed independently of other items in the list. This setting could prove sub-optimal for ranking problems for two main reasons. First, univariate functions have limited power to model cross-item comparison. Consider an *ad hoc* document retrieval scenario where a user is searching for the name of an artist. If all the results returned by the query (e.g., "calvin harris") are recent, the user may be interested in the latest news or tour information. If, on the other hand, most of the query results are older (e.g., "frank sinatra"), it is more likely that the user seeks information on artist discography or biography. Thus, the relevance of each document depends on the distribution of the whole list. Second, user interaction with search results shows a strong tendency to *compare* items. Prior research suggests that preference judgments by comparing a pair of documents are faster to obtain, and are more consistent than absolute ratings [38]. Moreover, it has been shown that better predictive capability is achieved when user actions are modeled in a relative fashion (e.g., SkipAbove) [4, 22]. These studies indicate that users compare a document with its surrounding documents prior

to a click, and that a ranking model that uses the direct comparison mechanism can be more effective, as it mimics user behavior more closely.

Given the above arguments, we hypothesize that the relevance score of an item should be computed by comparison with other items in the list at the feature level. Specifically, we explore a general setting of *multivariate* scoring functions for learning-to-rank. In its general form, a multivariate scoring function $f : \mathcal{X}^n \rightarrow \mathbb{R}^n$, where $\mathcal{X}$ is the universe of all items, takes a vector of n items as input and jointly maps them to an n-dimensional vector of reals. Each element in the output vector determines the *relative* relevance of an item with respect to other items in the input. While it is straightforward to define a multivariate function, it is less clear how such a function may be efficiently learned from training data or efficiently evaluated during inference given lists of arbitrary and variable number of items. To that end, we propose Groupwise Scoring Function (GSF) as an instance of the class of multivariate functions that is parameterized by deep neural networks. A GSF learns to score a fixed-size "group" of items. We show how this model can be extended to act on lists of arbitrary length and demonstrate how efficient training and inference can be achieved by a Monte Carlo sampling strategy. Empirical experiments on a private email search corpus and a public benchmark demonstrate that GSFs can achieve the state-of-the-art performance in learning-to-rank tasks.

In particular, our contributions can be summarized as follows:

- We motivate and formulate multivariate scoring functions for learning-to-rank;
- We present Groupwise Scoring Function (GSF) as an instance of the class of multivariate functions that is parameterized by deep neural networks;
- We explore the conditions under which a GSF reduces to existing learning-to-rank models;
- We demonstrate, through empirical evaluation on proprietary and public datasets, the improvements obtained by GSFs and discuss their potential for learning-to-rank tasks;
- To encourage research in this space and to allow for reproducibility of the reported results, we open source our implementation within the TF Ranking library [28].

2 RELATED WORK

Learning-to-rank refers to algorithms that model the ranking problem with machine learning techniques. In general, ranking is formulated as a score-and-sort problem with the objective of constructing a scoring function where scores computed by such a function induce an ordering of items in a list. Existing learning-to-rank algorithms [5, 6, 8, 14, 21, 35] mainly differ by two factors: (a) the parameterization of the scoring function (e.g., linear functions [21], boosted weak learners [37], gradient-boosted trees [6, 14], support vector machines [21, 23], and neural networks [5]); and (b) the loss function (e.g., pointwise [15], pairwise [5, 6, 21] and listwise [8, 35]). Virtually all of the existing algorithms, however, yield a univariate scoring function in the end where the score of an item is computed in isolation and independently of other items in the list. To the best of our knowledge, there are only a few exceptions.

First, the score regularization technique [12] and the CRF-based model [30] use document similarities to smooth the initial ranking scores or enrich query-document pair feature vectors. When computing relevance scores, however, both methods take only one document at a time.

The second exception is a bivariate scoring function [11] that takes a pair of documents as input and predicts the preference of one over the other. It is easy to show that the bivariate scoring function is a special case of our proposed framework.

Third is a group of neural learning to rank algorithms [2, 3] and click model [4] that builds an recurrent neural network over document lists. They, however, either focus on a re-ranking problem or use a pointwise loss to optimize user clicks. In contrast, our method can be applied to arbitrary number of documents with any types of ranking loss functions.

Search result diversification is another area of related work. Diversification algorithms maximize objectives that take subsets of documents into account. These include maximal marginal relevance [9] and subtopic relevance [1]. Recently, several deep learning algorithms were proposed with losses corresponding to those objectives [19, 36]. In contrast, our work focuses on improving relevance, not diversity, by way of cross-document comparisons.

Another area of related research is the work on pseudo-relevance feedback [25] where queries are expanded based on the top retrieved documents in a first round. The idea is that expanded queries lead to improvements in a second-stage retrieval. In this paper, we consider document relationships in the learning-to-rank setting, not retrieval, and do not require two rounds of retrieval. We also do not assume a pre-existing initial ordering of the document list.

Finally, note that our work is orthogonal and complementary to the recently proposed neural IR techniques [11, 16, 26, 27]. These techniques focus on advanced representations of document and query text but employ standard loss and scoring functions. On the other hand, our work concerns the nature of the scoring functions while employing a relatively simple query-doc representation.

3 PROBLEM FORMULATION

In this section, we formulate our problem in the context of learning-to-rank. Let $\psi = (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X}^n \times \mathbb{R}^n$ be a training sample where $\boldsymbol{x}$ is a vector of n items x_i, $1 \leq i \leq n$, $\boldsymbol{y}$ is a vector of real n relevance labels y_i, $1 \leq i \leq n$, and $\mathcal{X}$ is the space of all items. To simplify discussion and to follow convention, we refer to x_i simply as a "document" and $\boldsymbol{x} \in \mathcal{X}^n$ as a list of n documents, but note that x_i itself could be a feature vector representing a query-document pair. For every document $x_i \in \boldsymbol{x}$, we have a corresponding relevance label $y_i \in \boldsymbol{y}$. Finally, let Ψ be a set of training examples.

The goal of learning-to-rank can often be stated as finding a scoring function $f : \mathcal{X}^n \rightarrow \mathbb{R}^n$ that minimizes the empirical loss over the training data:

$$\mathcal{L}(f) = \frac{1}{|\Psi|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \Psi} \ell(\boldsymbol{y}, f(\boldsymbol{x})), \tag{1}$$

where $\ell(.)$ is a local loss function.

As noted in earlier sections, the main difference between the various learning-to-rank algorithms lies in how the scoring function $f(\cdot)$ and the loss function $\ell(\cdot)$ are defined. While there are numerous examples of prior work on different types of loss functions [24], the vast majority of learning-to-rank algorithms assume a *univariate* scoring function $u : \mathcal{X} \rightarrow \mathbb{R}$ that computes a score for

each document independently of other documents:

$$f(\boldsymbol{x})|_i = u(x_i),\ 1 \leq i \leq n, \tag{2}$$

where $f(\cdot)|_i$ denotes the i^{th} dimension of f.

A score obtained from $u(\cdot)$ depends only on its argument. In other words, fixing x_i and changing any or all other documents in the list to x' (for $j \neq i$) does not affect the output of $u(x_i)$.

In this paper, we set out to explore the space of *multivariate* scoring functions $f : \mathcal{X}^n \rightarrow \mathbb{R}^n$ for learning-to-rank. Following our discussion in Section 1, such a function is theoretically able to capture the relationship between its arguments and, as a result, could jointly produce *relative* scores. In other words, replacing x_i with a new document x_i' could lead to a change to scores for all documents in the list. Note, however, that any multivariate scoring function $f(\boldsymbol{x})$ should ideally be invariant to the order of items in $\boldsymbol{x}$.

Learning and evaluating a multivariate scoring function in practice is, however, nontrivial. In the discussion above, we made a simplifying assumption that n, the number of documents in a list, is constant across all training samples. As is common in learning-to-rank settings, however, that is often not the case and in fact the length of $\boldsymbol{x}$ is arbitrary and varies across training or evaluation samples. It is therefore not immediately clear how one may construct a generic multivariate function. In the following section, we address these challenges and introduce an instance of multivariate scoring functions that is suitable for the task of learning-to-rank and is further trained and evaluated in an efficient manner.

4 GROUPWISE SCORING FUNCTIONS

In this section, we present a detailed construction of an instance of multivariate scoring functions which we refer to as *groupwise scoring functions* (GSFs). A GSF in its basic form is a function $g(\cdot;\theta) : \mathcal{X}^m \rightarrow \mathbb{R}^m$ that is parameterized by a deep neural network (DNN) and that jointly maps a group of m documents (where m is fixed) to a vector of scores of the same size. We begin this section by laying out the foundations of a GSF, later proceed to extend it to lists of $n \geq m$ documents, where n may vary across samples, and finally complete the construction by providing a mechanism to efficiently train and evaluate an extended GSF.

4.1 Parameterization by DNNs

As noted earlier, we parameterize our functions using deep neural networks. Feed-forward neural networks have widely been applied to learning-to-rank problems [13, 17, 39]. The reasons we believe a deep neural network fits well into our framework are two-fold. First, compared to tree models, neural networks scale well to high-dimensional inputs. This is important because a GSF takes m documents as input where each document is a vector of an arbitrary and potentially large number of features. Second, neural networks arguably handle sparse features such as text more naturally whereas other models require extensive feature engineering. As such we believe a deep neural network is the right candidate for the task of learning a GSF.

To begin the construction, we need to define an input layer. Conceptually, a document x can be represented as a concatenation of two subsets of features: the embedding features for sparse textual features x^{embed} (e.g., for document titles) and the dense features

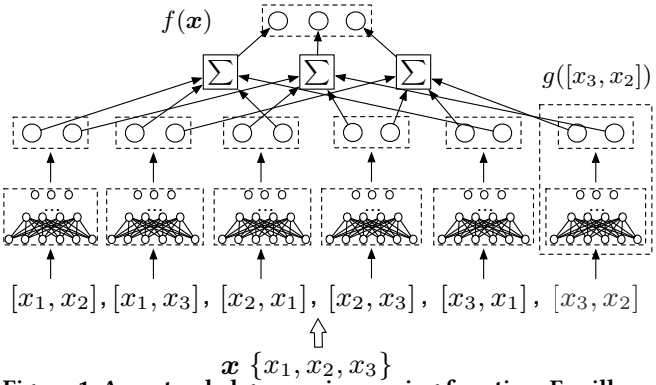

Figure 1: An extended groupwise scoring function. For illustrative purposes, we simplify $g(.)$ to be a bivariate function acting on permutations of size 2 formed from a list of 3 documents $\boldsymbol{x}$. All 6 size-2 permutations from $\boldsymbol{x}$ are fed to $g(.)$ which itself outputs 2 scores per permutation. Intermediate scores computed by g are subsequently aggregated to compute the final vector of scores f.

x^{dense} (e.g., document static scores or various match scores [16]). For simplicity, we construct the input layer by concatenating all m documents. Specifically, let

$$\boldsymbol{h}_0 = \text{concat}(x_1^{\text{embed}}, x_1^{\text{dense}}, ..., x_m^{\text{embed}}, x_m^{\text{dense}}).$$

Note that in practice the input layer can be extended to include document-independent "context" features (such as query embeddings) and need not be limited to document-derived features.

Given the above input layer, we build a multi-layer feed-forward network with 3 hidden layers as follows:

$$\boldsymbol{h}_k = \sigma(\boldsymbol{w}_k^T \boldsymbol{h}_{k-1} + \boldsymbol{b}_k),\ k = 1, 2, 3 \tag{3}$$

where $\boldsymbol{w}_k$ and $\boldsymbol{b}_k$ denote the weight matrix and the bias vector in the k-th layer, σ is an activation function, which in this work is the ReLU function: $\sigma(t) = \max(t, 0)$.

Our groupwise scoring function g is thus defined as:

$$g(\boldsymbol{x}) = \boldsymbol{w}_o^T \boldsymbol{h}_3 + \boldsymbol{b}_o \tag{4}$$

where $\boldsymbol{w}_o$ and $\boldsymbol{b}_o$ are the weight vector and the bias in the output layer. The output layer of the network consists of m units, each producing a score for each of the m documents.

We note that in this work we wish to keep the design of our input layer and network architecture simple as these details, while important and consequential, are not germane to the topic of this work. We leave the exploration of more sophisticated representation of groups of input documents and advanced layers as future work.

4.2 Extension to Arbitrarily Long Lists

The domain of the function $g(\cdot)$ presented in the previous section is $\mathcal{X}^m$ with m fixed. As noted earlier, in learning-to-rank, it is often the case that the list size (i.e., number of documents retrieved for a query) varies between queries. That important detail poses a challenge when designing and training a GSF.

Addressing this challenge by brute-force, one may set m to be the corpus size and subsequently zero-pad input lists during training and inference. To state the obvious, the resulting network clearly does not scale to real-world corpora. Moreover, given the enormity

of the parameter space, training such a network becomes prohibitive and any resulting model is unlikely to be effective.

A more viable solution, and one that we adopt to extend GSFs, is the following: Given a list of documents $\boldsymbol{x}$ of an arbitrary size n and a GSF $g : \mathcal{X}^m \rightarrow \mathbb{R}^m$, we propose to compute $g(.)$ on size-m permutations of $\boldsymbol{x}$ and accumulate scores along the way.

Let $\Pi_m(\boldsymbol{x})$ denote a set of all possible $\frac{n!}{(n-m)!}$ permutations of size m of the n documents in $\boldsymbol{x}$, and let $\pi_k \in \Pi_m(\boldsymbol{x})$ be an element of that set. A permutation π_k can be understood as a group of m documents. In our proposed method, we compute $g(\cdot)$ on π_k for all k. The vector of values $g(\pi_k)$ contains the scores of all documents $x_i \in \pi_k$ relative to other documents in that group. Group scores g are subsequently used to compute a final score for all n documents. To explain that, it helps to define the following function:

$$h(\pi, x) = \begin{cases} g(\pi)|_{\pi^{-1}(x)}, & \text{if } x \in \pi \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

where we use $\pi^{-1}(x)$ to denote the position of x in π. The final score $f(\cdot)$ is then calculated by the following equation:

$$f(\boldsymbol{x})|_i = \sum_{\pi_k \in \Pi_m(\boldsymbol{x})} h(\pi_k, x_i),\ 1 \leq i \leq n. \tag{6}$$

Figure 1 illustrates one such $f(\cdot)$ in a simplified setting where $g(\cdot)$ is bivariate and $\boldsymbol{x}$ is a list of 3 documents.

4.3 Efficient Training and Inference

One caveat of the extended GSF is the factorial growth of the space of permutations $\Pi_m(.)$. For large values of n, the set $\Pi_m(\boldsymbol{x})$ grows so intractably large that computing $g(.)$ on the resulting groups and aggregating group scores by Equation 6 quickly become prohibitive: assuming the computational complexity of $g(\cdot)$ is $O(m)$ such a scoring paradigm has a complexity of $O(m\frac{n!}{(n-m)!})$.

To reduce the complexity of GSFs, we propose to substitute the summation in Equation 6 with an expectation as follows:

$$f(\boldsymbol{x})|_i = \mathbb{E}_{\pi \ni x_i}[g(\pi, x_i)|_{\pi^{-1}(x_i)}],\ 1 \leq i \leq n. \tag{7}$$

The expectation in Equation (7) can be approximated effectively using Monte Carlo methods [31]. In our implementation, we use the following sampling recipe: From each training sample with document list $\boldsymbol{x}$, we form groups by taking sub-sequences of a randomly shuffled version of $\boldsymbol{x}$.

Such down-sampling substantially reduces the time complexity to $O(mn)$. It is easy to show that, because each $x_i \in \boldsymbol{x}$ appears in exactly m groups, each document is equally likely to be compared with other documents in the list. Moreover, a document's position in the group is also uniformly distributed. Given enough training data, a GSF trained using this sampling strategy asymptotically approaches a GSF trained with all permutations and is further invariant to document order in the input list.

4.4 Loss Function

We train a GSF by optimizing the empirical loss in Equation (1) using back-propagation. While in theory any arbitrary loss function $\ell(\cdot)$ can be used within this framework—more on this in Section 4.5—we empirically found the cross-entropy loss to be particularly effective. We define this loss as follows:

$$\ell(\boldsymbol{y}, f(\boldsymbol{x})) = -\sum_{i=1}^{n} \frac{y_i}{Y} \cdot \log p_i \tag{8}$$

where $Y = \sum_{y \in \boldsymbol{y}} y$ is a normalizing factor, and p_i's are the projection of scores $f(\boldsymbol{x})$ onto the probability simplex using Softmax:

$$Softmax(\boldsymbol{t})|_i = \frac{e^{t_i}}{\sum_{j=1}^{n} e^{t_j}},\ 1 \leq i \leq n. \tag{9}$$

An important property of this loss function is that it can be incorporated into an unbiased learning-to-rank framework. Specifically, it is easy to extend this loss to factor in Inverse Propensity Weights [23, 34] to counter position bias in click logs. The IPW-enabled variant of the loss in Equation (8) is as follows:

$$\ell(\boldsymbol{y}, f(\boldsymbol{x})) = -\sum_{i=1}^{n} w_i \cdot y_i \cdot \log p_i = -\sum_{i:y_i=1} w_i \cdot \log p_i, \tag{10}$$

where w_i is the Inverse Propensity Weight of the i^{th} result, and where it is assumed that $y \in \{0, 1\}$ for click logs and that only one document is clicked (i.e., $Y = 1$).

We use the above loss in the experiments reported in this work and leave the exploration of more advanced loss functions as future work. We will also defer a theoretical analysis of the cross-entropy loss or its extensions in the context of GSFs to a future study.

4.5 Relationship with Existing Models

In this section, we discuss the relationship between some of the existing learning-to-rank algorithms and our proposed model. In particular, a GSF model can be reduced to most existing algorithms by way of tuning a few knobs including group size m, loss function $\ell(\cdot)$, and the score aggregation function $f(\cdot)$. This includes RankNet [6], ListNet [8], and the work by Dehghani et al. [11]. Due to space limitation, we only show the ListNet as an example.

A traditional listwise model uses a univariate scoring function with a listwise loss that is computed over all documents in the list. It is easy to see how a GSF can be modified and reduced to a univariate function with a listwise loss: Fix $m = 1$ for $g(\cdot)$, define $f(\cdot)$ as in Equation (7), and plug any listwise loss $\ell(\cdot)$ into Equation (1).

Let us go through this exercise by presenting a configuration that transforms our GSF model to ListNet [8]. Given the univariate nature of $g(\cdot)$ in the new configuration, define $\hat{\boldsymbol{y}}$ as

$$\hat{\boldsymbol{y}} \triangleq f(\boldsymbol{x}),\ \hat{y}_i \triangleq f(\boldsymbol{x})|_i = g(x_i),\ 1 \leq i \leq n. \tag{11}$$

ListNet optimizes the cross-entropy loss between two ("top-one" probability) distributions: One obtained from relevance labels $\boldsymbol{y}$ and another defined over scores $\hat{\boldsymbol{y}}$. The following expression defines the ListNet loss:

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{n} \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \log \frac{e^{\hat{y}_i}}{\sum_{j=1}^{n} e^{\hat{y}_j}}. \tag{12}$$

Using the above loss in Equation (1) completes the transformation of a GSF to the ListNet model.

Note that the ListNet loss is almost identical to the loss used in our GSF model as shown in Equation (8). ListNet, however, projects labels to the probability simplex using the Softmax function whereas in GSF labels are simply normalized. When $y_i = 0$, we calculate

Table 1: List of baseline DNN models.

PointDNN	A standard DNN model with a univariate scoring function and pointwise loss [39].
RankNet	A neural network model with univariate scoring and pairwise loss [5].
BiDNN	The standard DNN model with bivariate scoring and Sigmoid cross entropy [11].

Table 2: List of GSF variants.

PairGSF	GSF reduced to a univariate scoring function with a pairwise loss used in RankNet [6].
BiGSF	GSF reduced to a bivariate scoring function similar to [11], but where the aggregation function remains as in Equation 7.
GSF(m)	GSF model with group size m.

zero loss in the GSF setup while this is not the case in the standard ListNet loss; the ListNet loss is always non-zero. This difference becomes important if one wishes to train a model in an unbiased learning-to-rank framework [23, 33] where propensity weights cannot be computed for non-clicked documents [34]. As such, having a non-zero loss for non-clicked documents proves to be a significant limitation of the ListNet loss in the context of unbiased learning.

5 EXPERIMENTAL SETUP

GSFs have theoretically interesting properties but their effectiveness in practice remains to be verified empirically. In the remainder of this paper, we set out to do just that by evaluating our proposed method on two datasets. To conduct experiments, we have implemented the GSF model in Tensorflow, a standard deep learning platform. In order to facilitate reproducibility of the reported results, we open source our code within the the TF Ranking library [28]. Moreover, in this section, we give a detailed description of our experimental design, setup, and model hyper-parameters.

5.1 Baseline Learning-to-Rank Models

We compare our method with a number of existing learning-to-rank algorithms that fall into two categories: DNN models and tree-based models. Table 1 summarizes a list of DNN models we use as baselines in our experiments. In this table, PointDNN and RankNet represent the existing DNN models with a univariate scoring function in the learning-to-rank literature. BiDNN is a recently proposed model that takes a pair of documents and jointly computes preference scores [11]. As for tree-based models, we primarily use the state-of-the-art MART and LambdaMART [6] algorithms as a baseline to compare with. In general, tree-based models cannot efficiently handle high-dimensional sparse features such as document text. Therefore, where we compare DNNs and tree-based models we do so by training a model with dense features only. Furthermore, where possible, we also explore a hybrid approach in which predictions from the DNN models are used as input features for tree-based models. Such a hybrid approach enables us to incorporate sparse features into tree-based models; we compare hybrid models with both standalone DNN and tree-based models in our experiments.

For completeness, Table 2 summarizes the different GSF variants considered in the following sections.

5.2 Datasets

We conduct a first set of experiments on a click dataset that is obtained from search logs of one of the largest commercial email search engines. In this service, a maximum of 6 results are returned and presented to users in an overlay. The overlay disappears after a click and the clicked result is then displayed. As a result, at most one click is obtained per query session. For this dataset, we discard all sessions that do not contain a click. For sessions with a click, we keep all 6 displayed documents and their click/no-click is recorded as relevance labels. This process results in approximately 150 million sessions in total. We sampled 5 million sessions to construct a held-out test set and used the rest for training and validation with a 9 : 1 ratio. To train BiDNN and BiGSF, we sample all pairs where one document is clicked.

The features in this dataset consist of both dense and sparse features. The dense features include query-document matching features like BM25. These types of features are the primary features used in traditional learning-to-rank algorithms [24]. Recently, sparse features were shown to be effective through embedding in an end-to-end deep neural network model [11]. Our click dataset contains n-grams from query strings and document subjects as sparse features. The average of the embedding vectors for n-grams in a query or document subject is used as the feature representation.

The second dataset used in our experiments is the publicly available MSLR-WEB30K [29]. This is a large-scale learning-to-rank set that contains 30,000 queries. On average there are 120 documents per query and each document has 136 numeric features. All documents are labeled with graded relevance from 0 to 4 with larger labels indicating a higher relevance. We evaluate the models on Fold 1 of this dataset. Results obtained on the other folds are similar.

5.3 Hyperparameters and Training

We build the DNN models using a 3-layer feed-forward network. On the Email dataset, hidden layer sizes are set as 256, 128 and 64 for $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ and $\boldsymbol{h}_3$ respectively. We set the learning rate to 0.1 and training batch size to 100. For sparse features, we set the embedding dimension to 20. Larger embedding dimensions (e.g., 100), learning rates (e.g., 0.2, 0.3), and layer sizes (e.g., 512 or 1024) were tested but no significant difference was observed. For this dataset, we use unbiased learning-to-rank techniques to overcome click bias [23, 34] and to that end, we optimize the weighted variant of the cross-entropy loss as shown in Equation (10) during training.

In models trained on Web30K, hidden layers have 64, 32, and 16 units instead with batch normalization between consecutive layers. A learning rate of 0.005 is used and training batch size is set to 128. These hyper-parameters were found to be effective through fine-tuning on the validation set. We train the model for 30,000 steps and evaluate the final model. Finally, when aggregating losses from queries in a mini-batch, a query's loss is weighted by the sum of the relevance grade of its documents.

For both datasets, we use Adagrad to optimize the objective.

5.4 Evaluation

For experiments on the Email dataset, we report an Inverse Propensity Weight (IPW) enabled variant of mean reciprocal rank (MRR) [33]. Such a weighted metric allows us to correct for the

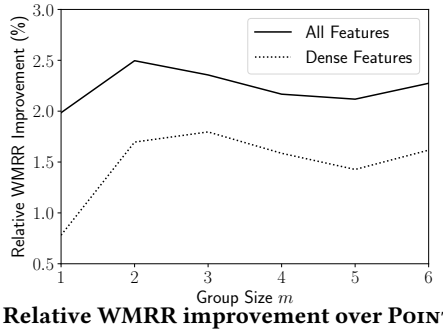

Figure 2: Relative WMRR improvement over PointDNN for GSF models with different group sizes using dense features and all (dense and sparse) features on the Email dataset.

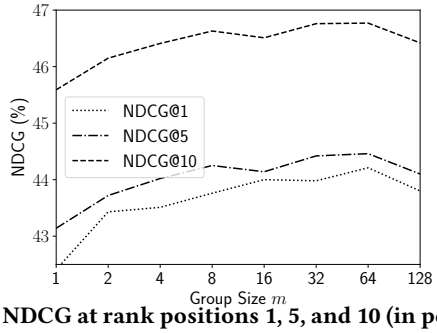

Figure 3: NDCG at rank positions 1, 5, and 10 (in percentage) for GSFs with different group sizes on the Web30K dataset.

position bias that exists in click logs. Let N denote the number of test sessions and $rank_i$ be the rank of the clicked document for the i^{th} session, then weighted MRR is calculated as follows:

$$WMRR = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{i=1}^{N} \frac{w_i}{rank_i} \tag{13}$$

where w_i is the IPW of the clicked document for the i^{th} session.

For experiments on the Web30K dataset, we run 10 trials of every model configuration and report mean Normalized Discounted Cumulative Gain [18] at rank positions 1, 5, and 10 along with 95% confidence intervals. Note that when computing NDCG, queries with no relevant documents are discarded from the evaluation set. Also, each trial may produce a different model given the same hyper-parameters due to the stochastic nature of network initialization, as well as batch-level and query-level shuffling of documents.

6 EXPERIMENTAL RESULTS

In this section, we report the results of our experiments. We first examine the effect of group sizes m on GSF models. We then compare GSFs with the state-of-the-art learning-to-rank algorithms.

6.1 Effect of Group Size

As discussed in Sections 4.2 and 4.3, while a GSF can easily and efficiently extend to a variable list size n, the group size m must be fixed before the construction of the model. To study the effect m has on resultant models, we conduct experiments with different configurations on both the Email and Web30K datasets.

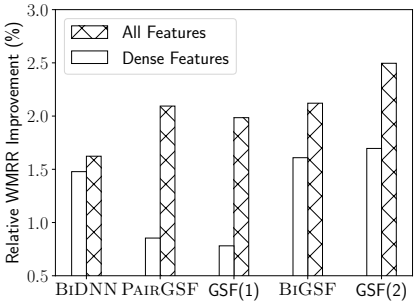

Figure 4: Relative WMRR improvements over PointDNN on all features and dense features on the Email dataset. All improvements over PointDNN are statistically significant according to a t-test with $\alpha < 0.01$. The improvement of GSF with group size 2, denoted GSF(2), over the other models is also statistically significant at $\alpha < 0.05$.

On the Email dataset, we train a PointDNN model (a univariate scoring function with pointwise loss, see Table 1) as baseline. We then measure improvements over this model, as indicated by WMRR, of GSF models trained with different group sizes. We repeat these experiments for two settings: (a) *all features* experiments use both sparse query and document textual features as well as numerical features; and, (b) *dense features* experiments use only the dense numerical features. Results are illustrated in Figure 2.

From Figure 2, we can see that a GSF trained with all features reaches its peak performance when $m = 2$, and GSF with dense features reaches its peak when $m = 3$. We also observe that the ranking quality decreases slightly when the group size becomes larger. We believe this observation can be explained by the fact that feed-forward networks usually are sensitive to the input order. As group size increases, the number of permutations for a group of documents grows rapidly. When this happens, because of the particular sampling process we use to form groups from a document list (see Section 4.3 for details), the approximation of the expectation in Equation (7) becomes less accurate.

On the Web30K dataset, we measure the performance of GSF models in terms of NDCG at rank positions 1, 5, and 10 and report these metrics for various group sizes. Figure 3 illustrates the results for groups of size 1, 2, 4, 8, 16, 32, 64, and 128. We observe an upward trend as we increase the group size m. The models with group size $m \geq 8$ are approximately 3% better than those with group size 1. Noting that there are only dense features in the Web30K dataset, this indicates that the extension of univariate scoring functions to multivariate scoring functions could be particularly useful for learning-to-rank models with dense features. Once again, for models with very large group sizes ($m \geq 32$), we observe that NDCG plateaus or drops slightly, which can be explained by inadequate sampling and the growth of the space of permutations.

6.2 Comparison with Baseline DNNs

We are interested in the relative performance of GSF models when compared with other baseline DNN algorithms. On the Email dataset, we measure the gain in WMRR over the PointDNN method obtained by PairGSF (univariate GSF with pairwise loss), BiGSF

Table 3: Relative WMRR improvement over LambdaMART on the Email dataset. $*$ and $+$ denotes statistically significant improvements over LambdaMART with dense features and all other models in the table, both using t-test with $\alpha < 0.01$.

	Dense Features	All Features
LambdaMART	0.00%	–
GSF(2)	0.30%*	2.40%*
LambdaMART+GSF(2)	0.95%*	**3.42%**$^{*+}$

(bivariate GSF), and finally GSF models with group sizes 1 and 2—we denote the last two models as GSF(1) and GSF(2) for brevity. To provide a reference point for how well GSF models perform, we also report the gain from the BiDNN model. The results of these comparisons on Email data is illustrated in Figure 4.

All five models achieve significant improvements over the PointDNN baseline. Among them, GSF(2) yields the highest WMRR for both "all" and "dense" feature settings. For example, using all features GSF(2) achieves a 2.5% improvement over the PointDNN baseline. This improvement is significantly better than the gain from BiDNN, an improvement of less than 1.5%.

If we consider dense features only, BiDNN, BiGSF, and GSF(2) models—all bivariate functions—lead to better results than GSF(1), a univariate function. This suggests that scoring documents jointly proves particularly effective when only dense features are available.

We also compared PairGSF, BiGSF, GSF(1) and GSF(2) in terms of NDCG@5 on the Web30k dataset. Results are shown in Table 4(a) which confirm again that the GSF is indeed more effective than the univariate and bivariate scoring functions.

6.3 Comparison with Tree-based Models

We next compare the proposed GSF models with tree-based models in both a standalone and a hybrid approach. In the hybrid setting—henceforth, referred to as LambdaMART+GSF—the output of the GSF model is used as a feature in LambdaMART. We use LambdaMART as reference because it has been shown to yield state-of-the-art performance in public learning-to-rank competitions [10].

Table 3 shows the results we obtained on the Email dataset. For scalability reasons, we use an internal implementation of LambdaMART on this dataset. As LambdaMART cannot natively handle raw textual features, we only report the relative improvement over LambdaMART with dense features. Based on the results in Figure 4, we use GSF(2) as the representative GSF model for this experiment.

From Table 3, we see that GSF significantly outperforms LambdaMART in (a) dense features regime (where GSF slightly outperforms LambdaMART), and (b) all features regime (where the performance gap is much more significant). This demonstrates the importance of incorporating raw textual features and the effectiveness of GSF models in leveraging them. Furthermore, the hybrid LambdaMART+GSF approach achieves an even better performance, reaching gains as large as 3.42% over LambdaMART, a statistically significant improvement over all other models in Table 3. This validates the complementary nature of our method to LambdaMART and the benefits of the hybrid approach.

Table 4 shows the results on the Web30K dataset. For reproducibility, we use several learning-to-rank models implemented in the open-source Ranklib toolkit[1] as baselines.

[1]https://sourceforge.net/p/lemur/wiki/RankLib/

Table 4: A comparison on the Web30K dataset of (a) various GSF flavors and weaker baselines by NDCG@5; (b) strong baseline models and the best-performing GSF variant by NDCG at different cut-offs with 95% confidence intervals from 10 trials; and, (c) highest performing trial as measured by NDCG at different rank positions on the validation set. * denotes statistically significant differences between GSF and LambdaMART using t-test with $\alpha < 0.05$.

(a)

RankNet	RankSVM	PairGSF	BiGSF	GSF(1)	GSF(2)	GSF(64)
32.28	34.79	40.40	41.10	43.14	43.72	**44.46**

(b)

	MART	LambdaMART	GSF(64)
NDCG@1	43.73 (±0.01)	**45.35** (±0.06)	44.21 (±0.18)
NDCG@5	43.96 (±0.03)	**44.59** (±0.04)	44.46 (±0.12)
NDCG@10	46.40 (±0.02)	46.46 (±0.03)	**46.77** (±0.13)

(c)

	MART	LambdaMART	GSF(64)
NDCG@1	43.76	**45.27***	44.47
NDCG@5	44.03	44.56	**44.63**
NDCG@10	46.44	46.52	**47.01***

We observe that all GSF variants in Table 4(a), outperform RankNet and RankSVM by a very large margin. A comparison of GSFs with MART and LambdaMART is shown in Table 4(b), where we report mean NDCG at various rank positions over 10 trials along with 95% confidence intervals. From the table, it is clear that the GSF setting with group size $m = 64$ yields statistically significant improvements over MART at all NDCG cut-offs. On the other hand, GSF(64) falls short of LambdaMART as measured by NDCG@1, is on par in terms of NDCG@5 (i.e., confidence intervals overlap), and performs significantly better than LambdaMART as indicated by NDCG@10. For completeness, we select the trial with the highest NDCG@1 on the validation set and measure its NDCG@1 on the test set. We repeat this for NDCG@5 and NDCG@10 and report the results in Table 4(c). The conclusions from Table 4(b) still hold.

The results from Table 4 are interesting. We believe the reason LambdaMART performs better than GSF at NDCG@1 is a result of the differences between loss functions: in the existing GSF setup, we use the cross-entropy loss which is not position-dependent, whereas LambdaMART's loss is designed to take position into account.

Our observations from a comparison of GSFs with tree-based models lead us to believe that the ranking quality of GSFs is at least on par with state-of-the-art tree-based models. As the number of training examples increases by orders of magnitude and when sparse textual features are present in a feature set, GSFs yield higher quality models and prove more scalable.

7 DISCUSSION AND FUTURE WORK

We began this work by stating a hypothesis, that the *relative* relevance of an item would be more accurately estimated if relevance scores for all items were computed jointly. Experiments conducted in the last section shed light on that hypothesis and the questions raised earlier in this work. The results are encouraging.

GSFs, while not yet a fully mature deep learning framework, provide a blueprint for designing multivariate scoring functions for ranking. Analogous to the use of Recurrent Neural Networks in Natural Language Processing and Convolutional Neural Networks in Computer Vision, we believe GSFs are inspired by and are

more appropriate for ranking, where relativity plays a large role. Moreover, GSFs incorporate local feature distributions by simultaneously considering multiple candidate documents, mimicking user behavior more closely. Thus, we believe that GSFs provide an opportunity for the advancement of learning-to-rank research using deep learning.

There are, for example, many components that warrant a closer look. A naïve concatenation of a list of input documents, as done in this work, may not be effective at preserving documents' structure and may lead to a loss of signals useful for comparison of documents. A feed-forward network may not be appropriate for capturing similarities or differences between documents. Finally, while the cross-entropy loss proved effective in practice, (a) it lacks theoretical justification and (b) a metric-driven loss may lead to better overall performance. We plan to pursue this direction of research and continue to improve our understanding of multivariate scoring and ranking functions. In order to facilitate and share this research, we open source GSFs within the TF Ranking library [28].

8 ACKNOWLEDGMENTS

This work would not be possible without the support provided by the TF-Ranking team.

REFERENCES

[1] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. 2009. Diversifying Search Results. In *Proc. of the 2nd ACM International Conference on Web Search and Data Mining*. 5–14.

[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 135–144.

[3] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. 2018. Seq2slate: Re-ranking and slate optimization with rnns. *arXiv preprint arXiv:1810.02019* (2018).

[4] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *Proc. of the 25th International Conference on World Wide Web*. 531–541.

[5] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proc. of the 22nd International Conference on Machine Learning*. 89–96.

[6] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report Technical Report MSR-TR-2010-82. Microsoft Research.

[7] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. 2006. Learning to Rank with Nonsmooth Cost Functions. In *Proc. of the 19th International Conference on Neural Information Processing Systems*. 193–200.

[8] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proc. of the 24th International Conference on Machine Learning*. 129–136.

[9] Jaime Carbonell and Jade Goldstein. 1998. The Use of MMR, Diversity-based Reranking for Reordering Documents and Producing Summaries. In *Proc. of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 335–336.

[10] O. Chapelle and Y. Chang. 2011. Yahoo! Learning to Rank Challenge Overview. In *Proc. of the Learning to Rank Challenge*. 1–24.

[11] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *Proc. of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 65–74.

[12] Fernando Diaz. 2007. Regularizing query-based retrieval scores. *Information Retrieval* 10, 6 (2007), 531–562.

[13] Bora Edizel, Amin Mantrach, and Xiao Bai. 2017. Deep Character-Level Click-Through Rate Prediction for Sponsored Search. In *Proc. of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 305–314.

[14] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[15] Fredric C. Gey. 1994. Inferring Probability of Relevance Using the Method of Logistic Regression. In *Proc. of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 222–231.

[16] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *Proc. of the 25rd ACM International Conference on Information and Knowledge Management*. 55–64.

[17] 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 *Proc. of the 22nd ACM International Conference on Information and Knowledge Management*. 2333–2338.

[18] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.

[19] Zhengbao Jiang, Ji-Rong Wen, Zhicheng Dou, Wayne Xin Zhao, Jian-Yun Nie, and Ming Yue. 2017. Learning to Diversify Search Results via Subtopic Attention. In *Proc. of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 545–554.

[20] Thorsten Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 133–142.

[21] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 217–226.

[22] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately Interpreting Clickthrough Data As Implicit Feedback. In *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 154–161.

[23] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *Proc. of the 10th ACM International Conference on Web Search and Data Mining*. 781–789.

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

[25] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[26] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *Proc. of the 26th International Conference on World Wide Web*. 1291–1299.

[27] 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 *Proc. of the 2017 ACM Conference on Information and Knowledge Management*. 257–266.

[28] Rama Kumar Pasumarthi, Xuanhui Wang, Cheng Li, Sebastian Bruch, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2018. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. (2018). arXiv:arXiv:1812.00073

[29] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. (2013). arXiv:1306.2597

[30] Tao Qin, Tie-Yan Liu, Xu-Dong Zhang, De-Sheng Wang, and Hang Li. 2008. Global ranking using continuous conditional random fields. In *Proc. of the 21st International Conference on Neural Information Processing Systems*. 1281–1288.

[31] Christian P. Robert and George Casella. 2005. *Monte Carlo Statistical Methods*. Springer-Verlag.

[32] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. SoftRank: Optimizing Non-smooth Rank Metrics. In *Proc. of the 1st International Conference on Web Search and Data Mining*. 77–86.

[33] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *Proc. of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 115–124.

[34] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *Proc. of the 11th International Conference on Web Search and Data Mining*. 610 –618.

[35] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proc. of the 25th International Conference on Machine Learning*. 1192–1199.

[36] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2016. Modeling Document Novelty with Neural Tensor Network for Search Result Diversification. In *Proc. of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 395–404.

[37] Jun Xu and Hang Li. 2007. AdaRank: A Boosting Algorithm for Information Retrieval. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 391–398.

[38] Peng Ye and David Doermann. 2013. Combining preference and absolute judgements in a crowd-sourced setting. In *ICML 2013 Workshop on Machine Learning Meets Crowdsourcing*.

[39] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational Context for Ranking in Personal Search. In *Proc. of the 26th International Conference on World Wide Web*. 1531–1540.