Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks

Sebastian Bruch, Masrour Zoghi, Michael Bendersky, Marc Najork
Google Research
{bruch,mzoghi,bemike,najork}@google.com

ABSTRACT
Learning-to-Rank is a branch of supervised machine learning that seeks to produce an ordering of a list of items such that the utility of the ranked list is maximized. Unlike most machine learning techniques, however, the objective cannot be directly optimized using gradient descent methods as it is either discontinuous or flat everywhere. As such, learning-to-rank methods often optimize a loss function that either is loosely related to or upper-bounds a ranking utility instead. A notable exception is the approximation framework originally proposed by Qin et al. [14] that facilitates a more direct approach to ranking metric optimization. We revisit that framework almost a decade later in light of recent advances in neural networks and demonstrate its superiority empirically. Through this study, we hope to show that the ideas from that work are more relevant than ever and can lay the foundation of learning-to-rank research in the age of deep neural networks.

CCS CONCEPTS
• Information systems → Learning to rank;

KEYWORDS
Direct Ranking Metric Optimization; Deep Neural Networks for IR; Learning to Rank

ACM Reference Format:

1 INTRODUCTION
Learning to rank (LTR) is a central problem in information retrieval (IR), where the task is to devise a ranking scheme that reorders a list of retrieved documents in response to a query such that the most relevant results appear as close to the top of the list as possible. In order to measure the quality of such ranked lists, many ranking metrics have been proposed, including Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP).

Given the LTR task and the corresponding evaluation metrics, the first proposal would be to train a LTR model by directly optimizing a metric like NDCG. As explained in Section 2, it is known, however, that ranking metrics including NDCG are non-differentiable and therefore impossible to optimize using gradient descent methods. Moreover, in the regions where the metrics are smooth, infinitesimal perturbations of our model parameters will almost surely leave the ranked list unperturbed, which in turn implies that whatever gradients we compute will be identically zero almost everywhere. Faced with this stumbling block, the LTR community has produced a plethora of schemes to improve metrics like NDCG, including metric smoothing methods such as SoftRank [15] and indirect boosting methods like LambdaMART [17].

A more direct approach to LTR metric optimization was proposed by Qin et al. [14], where the rank variable in the definition of metrics like NDCG was approximated by a sum of sigmoids, thereby allowing for gradient computations. However, this idea happened to be proposed at a time when tree-based LTR models were making great strides, as demonstrated for instance by LambdaMART’s winning of the Yahoo! Learning to Rank Challenge [5], and since regression trees cannot be optimized globally, such differentiable approximations of the metrics offered no immediate advantage. Recent hardware and software advances in the training of neural networks, however, make the work in [14] relevant again and potentially allow us to harvest the effectiveness and the scalability of deep neural networks in LTR.

In this paper, we make the following contribution: we demonstrate that directly optimizing NDCG, rather than a surrogate loss, using deep neural networks can give results that are comparable with those obtained using existing state-of-the-art LTR algorithms such as LambdaMART. We give an overview of LTR and in particular [14] in Section 2. We discuss experimental results in Section 3 and conclude the paper in Section 4.

2 RELATED WORK AND METHODOLOGY
In this section, we formulate the problem of LTR and provide an overview of the literature. We also provide a self-contained summary of the core idea behind the ApproxNDCG [14] method.

2.1 Overview of Learning-to-Rank
LTR methods are supervised techniques and the story naturally begins with a description of the training set. Consider a set of training samples \( \Psi = \{(x_i, y_i) \in \mathcal{X}^n \times \mathbb{R}^n \} \), where \( x \) is a vector of \( n \) items \( x_i, 1 \leq i \leq n \), \( y \) is a real vector of \( n \) nonnegative relevance labels \( y_i, 1 \leq i \leq n \), and \( \mathcal{X} \) is the space of all items. Each item \( x \) could generally take any form but throughout this paper we define it to be a vector of features representing a query-document pair. The objective is to learn a function that produces an ordering of
items in any $x$ in such a way that the utility of the ordered list is maximized.

Most LTR algorithms reformulate the problem to that of learning a scoring function that computes a score for every item. Scores usually represent relevance—for some notion of relevance—and induce an ordering of the items by sorting them in decreasing order of relevance to form a ranked list. As such, the goal of LTR often boils down to finding a parameterized ranking function $f(\cdot; \Theta) : X^n \rightarrow \mathbb{R}^n$, where $\Theta$ denotes the set of parameters, that minimizes the empirical loss:

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

where $\ell(\cdot)$ is a local loss function. The function $f$ is often univariate and can be rewritten as follows:

$$
f(x)_i = u(x)_i, \ 1 \leq i \leq n,
$$

where $f(\cdot)_i$ denotes the $i^{th}$ dimension of $f$, and $u : X \rightarrow \mathbb{R}$ computes a relevance score for each item independently of other items.

LTR algorithms differ primarily in how they parameterize $f$ and how they define $\ell$. Tried and tested parameterization methods include linear functions [9], boosted weak learners [19], gradient-boosted trees [3, 6], support vector machines [9], and neural networks [2]. In this paper, we model $f$ using the latter.

The loss function, $\ell$, is ideally derived from a utility of interest such as NDCG [8], a popular ranking metric. However, most ranking metrics are either discontinuous or flat. Take NDCG as an example:

$$
\text{NDCG}(\pi f, y) = \frac{\text{DCG}(\pi f, y)}{\text{DCG}(\pi^*, y)},
$$

where $\pi f$ is a ranked list induced by the ranking function $f$ on $x$, $\pi^*$ is the ideal ranked list (where $x$ is sorted by $y$), and DCG is defined as follows:

$$
\text{DCG}(\pi, y) = \sum_{j=1}^n \frac{2^{y_j} - 1}{\log_2(1 + j)},
$$

where $\pi(i)$ is the rank of $x_i$. In Eq. 4, small perturbations of the scores would not change the ranks for generic scores, and therefore NDCG is locally constant almost everywhere. Also, when the item ranks do change, NDCG becomes discontinuous.

The non-differentiability of ranking metrics has given rise to a body of research that attempts to find differentiable surrogate losses that either are loosely related to or upper-bound ranking metrics [2–4, 9, 18]. There exist a few notable exceptions that attempt to directly maximize a ranking metric by using coordinate ascent [11], smoothing scores [15], boosting [19], and approximating the metric [14]. It is the latter that can tightly bound any ranking metric such as NDCG [14] and can be easily optimized with gradient descent.

Surprisingly, despite its attractive theoretical properties, the framework in [14] has received little attention in LTR studies in the decade since the original publication. In this paper, we revisit the work that led to recent advances in deep neural networks and the availability of powerful optimizers. With significantly more computing power at our disposal today, we set out to study the hyperparameters of that work and reproduce experiments to optimize NDCG—referred to as ApproxNDCG. Our results show that the theoretical guarantees in [14] materialize in practice. Before we go any further, we give a brief overview of ApproxNDCG in the next section for completeness.

### 2.2 Summary of ApproxNDCG

As shown in Equation 4, to compute DCG, all that is required is the rank of items in the final ranked list as ordered by relevance scores. Moreover, the rank of an item $i$ can be computed as follows:

$$
\pi_f(i) = 1 + \sum_{j \leq i} I_{f(x)_j < f(x)_i},
$$

where $f(\cdot)$ is the scoring function from Equation 2, and $I_{t < s}$ is the indicator which is 1 if $s < t$ and 0 otherwise.

Qin et al. propose in [14] a smooth approximation of Equation 5 where $I$ is estimated by a sigmoid as follows:

$$
\hat{I}_{s < t} = \frac{1}{1 + e^{-\alpha(t-s)}},
$$

where $\alpha > 0$ is a knob that controls how tightly the sigmoid fits the indicator. As $\alpha$ becomes larger, $\sigma$ approximates the indicator more closely as shown in Figure 1.

Unlike the indicator function, the approximation in Equation 6 is smooth and differentiable. Plugging this approximation into Equation 4 yields ApproxNDCG, an approximation of NDCG. Because NDCG is a utility, we define the loss $\ell$ in Equation 1 to be negative ApproxNDCG and minimize the loss using gradient descent.

### 3 EXPERIMENTS

We are largely interested in two research questions alluded to earlier: (1) What is the impact of the hyperparameter $\alpha$ on the learned model? (2) Can directly optimizing the ranking metric with deeper networks and a much larger number of training iterations lead to higher quality models? In this section, we describe the experiments we designed to study those questions and analyze the results.

### 3.1 Datasets

We conduct exhaustive experiments on two publicly available LTR datasets: MSLR-WEB30K [13] and Yahoo! LTR Challenge [5]. Both datasets contain roughly 30,000 queries. Web30K has an average of 120 documents per query, each represented by a vector of 136 numeric features. Yahoo! Set 1 has 24 documents per query and 519 features per document. Documents in both datasets are labeled with graded relevance from 0 to 4 with larger labels indicating a higher relevance. We report our findings on Fold 1 of Web30K and Set 1 of the Yahoo! dataset. It is important to note that in both datasets, queries with no relevant documents are discarded during evaluation.
We describe the architecture of our networks in more detail in upcoming sections. As stated earlier, the first factor we examine in this work is the effect of \( \alpha \) on NDCG at different ranks on the Web30K validation set.

### 3.2 Models

We have compared our results with existing ranking models including ListMLE [18], RankNet and LambdaMART [3]. To train LambdaMART models, we used the recent open-source LightGBM [10] implementation (denoted by \( \lambda \text{MART}_{\text{GBM}} \)). We also used the legacy RankLib implementation (\( \lambda \text{MART}_{\text{RankLib}} \)). We implemented ListMLE and RankNet in Tensorflow [1], a deep learning framework. In all of our experiments, we run 10 trials of each experiment and report mean metrics and 95% confidence intervals.

The hyperparameters for LambdaMART models are based on those reported in previous work (e.g., [16]) and further fine-tuned on the validation set. Specifically, we train \( \lambda \text{MART}_{\text{RankLib}} \) models by setting the hyperparameter values as follows: number of leaves per tree to 10, learning rate to 0.1, minimum leaf support to 1. We were unable to train larger trees as larger parameter settings lead to a substantial and prohibitive rise in memory usage and training time. LightGBM, on the other hand, is an efficient implementation and as such we set the hyperparameters for \( \lambda \text{MART}_{\text{GBM}} \) as follows: learning rate is 0.1, number of leaves is 200, min data in leaves is 50, and min sum hessian in leaf is set to 100. We use NDCG@5 as the main metric to select the best models on validation sets.

Our proposed method is a fully-connected feedforward network with ReLU activation (ReLU(\( t \)) = max(0, \( t \))) using ApproxNDCG as the loss function; henceforth, we will also use ApproxNDCG to refer to this type of model. The models are trained as follows: similar to baseline models, the hyperparameters of the ApproxNDCG models are selected based on NDCG@5 on the validation set; training batch size is set to 128; and we use a learning rate of 0.005. We further use batch normalization [7] between consecutive layers, including over the input layer to, in effect, normalize input features. We describe the architecture of our networks in more detail in upcoming sections.

We have released our implementation of ApproxNDCG in Tensorflow in the open-source Tensorflow Ranking library [12].¹

### 3.3 Effect of the Sigmoid Exponent

As stated earlier, the first factor we examine in this work is the effect of \( \alpha \) in Equation 6 on the trained model. To that end, we train networks with a single hidden layer to limit the parameter space, but use different values of \( \alpha \). Figure 2 illustrates the results on Web30K. Experiments on Yahoo! yield a similar trend.

The results are interesting but not surprising. When \( \alpha \) is small, the sigmoid approximates the indicator function less accurately as shown in Figure 1. As such, the model optimizes a loss that is only loosely related to NDCG. On the other hand, when \( \alpha \) is too large, the sigmoid becomes flatter closer to the origin. As a result, the gradients tend to vanish which impedes learning. It appears, however, that a relatively small value of \( \alpha \) offers a compromise between the two extremes: it is a close-enough approximation of the indicator while also enabling gradient descent.

Note that the difference between models trained using \( \alpha \in \{5, 10, 20, 40\} \) is not statistically significant. We choose \( \alpha = 10 \) as the configuration of choice in the remainder of this paper purely based on its relatively superior NDCG@5 on the validation set.

### 3.4 Effect of Deeper Networks

Now that we have found an optimal value for \( \alpha \), we focus on the second question to study the effect of deeper networks on model quality. We start with a small network with 3 hidden layers with 64, 32, and 16 hidden units each. We refer to that as B64. We then construct progressively deeper models by adding layers that grow by a factor of 2. As an example, B128 will have 128 units in the first hidden layer, and 64, 32, and 16 units in subsequent layers. As stated earlier, layers are fully connected with batch normalization and ReLU nonlinearity in between.

Figure 3 plots model quality as measured by NDCG at various ranks on the Web30K validation set. Results on Yahoo! exhibit a similar trend. From Figure 3, it is clear that deeper models generally lead to improved quality. We note that the differences in NDCG@1 are not statistically significant, but that adding more and wider layers yield NDCG@5 and NDCG@10 measurements that statistically significantly improve upon shallower networks. The largest network exhibits signs of overfitting but we note that no regularization was employed in these experiments.

### 3.5 Comparison with Baseline Models

Based on the experiments conducted in previous sections, we use the B1024 model with \( \alpha = 10 \) and compare its performance with

---

¹Available at http://github.com/tensorflow/ranking
baseline methods on both Web30K and Yahoo! held-out test sets. We further fine-tune the "momentum" hyperparameter of batch normalization—used in the estimation of population statistics—and set it to 0.8 and 0.99 in the Web30K and Yahoo! experiments respectively. We use the same network architecture and hyperparameters for ListMLE and RankNet methods to facilitate a fair comparison. Table 1 summarizes our findings. From Table 1, we observe that ApproxNDCG significantly outperforms MART, on both datasets, but does not do as well as MART. Note that our NDCG measurements for MART are lower than those reported in previous work (e.g., [16]). This is because LightGBM computes an NDCG of 1.0 for queries with no relevant documents. In this work, we exclude such queries from the evaluation set to facilitate a fair comparison of scores. By comparing ApproxNDCG with ListMLE and RankNet, we conclude that the success of ApproxNDCG is not simply due to the use of deeper networks: the loss function itself is a more appropriate choice than the losses used in RankNet or ListMLE. We omit RankNet and ListMLE results on Yahoo! due to space constraints, but the findings are similar.

4 DISCUSSION AND FUTURE WORK

Deep neural networks have enabled a significant leap forward in many applications of machine learning such as NLP and Image Processing. Our ability to train scalable deep networks that handle sparse features such as text are among the factors that place neural networks at the vanguard of machine learning research. Harvesting these abilities in LTR, however, remains a challenge due to the discontinuous nature of ranking utility functions.

In this work, we set out to revisit the work of Qin et al. [14] which formulates a smooth approximation to any ranking metric such as NDCG. Unlike many other existing surrogate LTR losses, the framework in [14] offers a way to directly optimize ranking metrics. Because the objective is differentiable, it is also a good fit for gradient descent algorithms.

We studied ApproxNDCG, an approximation to NDCG, and examined its hyperparameter. We demonstrated empirically that ApproxNDCG greatly benefits from deep network architectures and, despite the little attention it received in the LTR literature, is a competitive algorithm for ranking.

Through this study, we hope to convey that (a) it is not just plausible but more appropriate to directly optimize ranking metrics rather than loosely related surrogate losses; and (b) that the approximation framework in [14] could lay out the foundation of deep neural networks in LTR. We wish to encourage research in this direction by open sourcing our implementation of ApproxNDCG in the TensorFlow Ranking library.

5 ACKNOWLEDGEMENTS

This work would not be possible without the support provided by the TF-Ranking team. We thank Donald Metzler for his input on an early draft of this work, and the anonymous reviewers for their feedback. The first author’s deepest gratitude goes to Katherine for her invaluable encouragement and wholehearted support.

REFERENCES


