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:
Sebastian Bruch, Masrour Zoghi, Michael Bendersky, Marc Najork. 2019. Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3331184.3331347

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).

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07.
https://doi.org/10.1145/3331184.3331347

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 $\Psi = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X}^n \times \mathbb{R}_+^n\}$, where $\boldsymbol{x}$ is a vector of n items x_i, $1 \leq i \leq n$, $\boldsymbol{y}$ 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_i 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 $\boldsymbol{x}$ 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 $f(\cdot;\Theta): \mathcal{X}^n \rightarrow \mathbb{R}^n$, where Θ denotes the set of parameters, that minimizes the empirical loss:

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

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

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

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

LTR algorithms differ primarily in how they parameterize f and how they define ℓ. 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, ℓ, 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:

$$NDCG(\pi_f, \boldsymbol{y}) = \frac{DCG(\pi_f, \boldsymbol{y})}{DCG(\pi^*, \boldsymbol{y})}, \tag{3}$$

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

$$DCG(\pi, \boldsymbol{y}) = \sum_{i=1}^{n} \frac{2^{y_i} - 1}{\log_2(1 + \pi(i))}, \tag{4}$$

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 that work in light of 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.

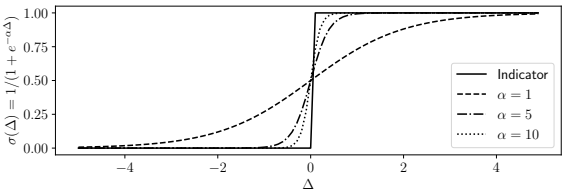

Figure 1: Sigmoid approximation of the indicator function with different values of hyperparameter α.

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:

$$\pi_f(i) \triangleq 1 + \sum_{j\neq i} \mathbb{I}_{f(\boldsymbol{x})|_i < f(\boldsymbol{x})|_j}, \tag{5}$$

where $f(\cdot)$ is the scoring function from Equation 2, and $\mathbb{I}_{s<t}$ 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 $\mathbb{I}$ is estimated by a sigmoid as follows:

$$\mathbb{I}_{s<t} = \mathbb{I}_{t-s>0} \approx \sigma(t-s) \triangleq \frac{1}{1+e^{-\alpha(t-s)}}, \tag{6}$$

where $\alpha > 0$ is a knob that controls how tightly the sigmoid fits the indicator. As α becomes larger, σ 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 ℓ 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 α 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.

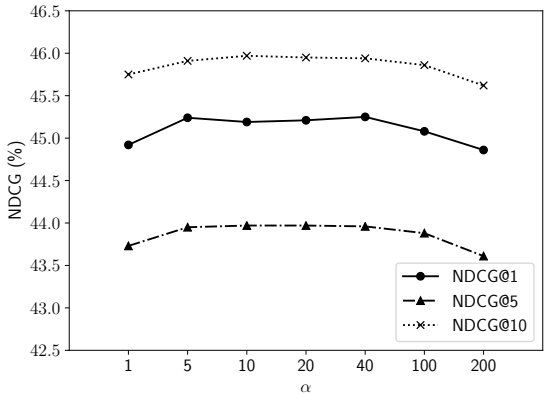

Figure 2: Effect of the sigmoid exponent, α, on NDCG at different ranks on the Web30K validation set.

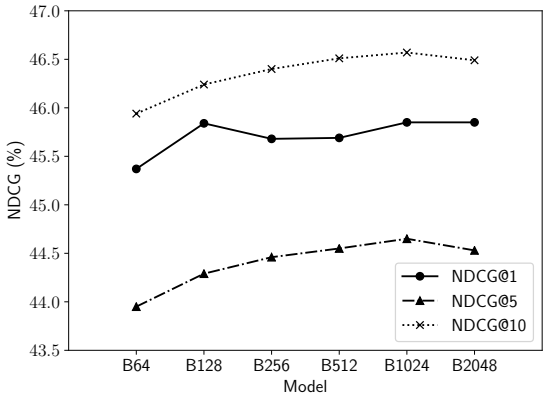

Figure 3: Effect of network depth on quality as measured by NDCG at different ranks on the Web30K validation set.

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 $\lambda\text{MART}_{\texttt{GBM}}$). We also used the legacy RankLib implementation ($\lambda\text{MART}_{\texttt{RankLib}}$). 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 $\lambda\text{MART}_{\texttt{RankLib}}$ 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 $\lambda\text{MART}_{\texttt{GBM}}$ as follows: learning rate is 0.1, number of leaves is 200, min_data_in_leaf 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 ($\text{ReLU}(t) = \max(t, 0)$) 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 α 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 α. Figure 2 illustrates the results on Web30K. Experiments on Yahoo! yield a similar trend.

The results are interesting but not surprising. When α 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 α 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 α 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 α, 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

[1] Available at http://github.com/tensorflow/ranking

Table 1: A comparison of ranking models on the Web30K and Yahoo! test sets.

Model	NDCG@1	NDCG@5	NDCG@10
	Web30K		
ListMLE	41.90 (±0.34)	42.56 (±0.20)	44.91 (±0.17)
RankNet	42.18 (±0.35)	43.23 (±0.14)	45.70 (±0.10)
λMART$_{\texttt{RankLib}}$	45.35 (±0.06)	44.59 (±0.04)	46.46 (±0.03)
λMART$_{\texttt{GBM}}$	**50.33** (±0.22)	**49.20** (±0.07)	**51.05** (±0.02)
ApproxNDCG	46.64 (±0.22)	45.38 (±0.11)	47.31 (±0.10)
	Yahoo! Set 1		
λMART$_{\texttt{RankLib}}$	68.52 (±0.09)	70.27 (±0.05)	74.58 (±0.05)
λMART$_{\texttt{GBM}}$	**72.07** (±0.22)	**74.16** (±0.14)	**78.40** (±0.10)
ApproxNDCG	69.63 (±0.17)	72.32 (±0.10)	76.77 (±0.06)

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 λMART$_{\texttt{RankLib}}$ on both datasets, but does not do as well as λMART$_{\texttt{GBM}}$. Note that our NDCG measurements for λMART$_{\texttt{GBM}}$ 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

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning. In *Proc. of the 12th USENIX Symposium on Operating Systems Design and Implementation*. 265–283.

[2] 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.

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

[4] 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.

[5] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proc. of the Learning to Rank Challenge*. 1–24.

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

[7] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. of the 32nd International Conference on Machine Learning (ICML)*. 448–456.

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

[9] 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.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*. 3146–3154.

[11] Donald A Metzler, W Bruce Croft, and Andrew Mccallum. 2005. *Direct maximization of rank-based metrics for information retrieval*. CIIR report 429. University of Massachusetts.

[12] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. (to appear).

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

[14] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval* 13, 4 (2010), 375–397.

[15] 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.

[16] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *Proc. of the 27th ACM International Conference on Information and Knowledge Management*. 1313–1322.

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

[18] 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.

[19] 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.