NeuralNDCG: Direct Optimisation of a Ranking Metric via Differentiable Relaxation of Sorting

Przemysław Pobrotyn[*]
ML Research at Allegro.pl
przemyslaw.pobrotyn@allegro.pl

Radosław Białobrzeski[*] [†]
ML Research at Allegro.pl
radoslaw.bialobrzeski@allegro.pl

ABSTRACT

Learning to Rank (LTR) algorithms are usually evaluated using Information Retrieval metrics like Normalised Discounted Cumulative Gain (NDCG) or Mean Average Precision. As these metrics rely on sorting predicted items' scores (and thus, on items' ranks), their derivatives are either undefined or zero everywhere. This makes them unsuitable for gradient-based optimisation, which is the usual method of learning appropriate scoring functions. Commonly used LTR loss functions are only loosely related to the evaluation metrics, causing a mismatch between the optimisation objective and the evaluation criterion. In this paper, we address this mismatch by proposing NeuralNDCG, a novel differentiable approximation to NDCG. Since NDCG relies on the non-differentiable sorting operator, we obtain NeuralNDCG by relaxing that operator using NeuralSort, a differentiable approximation of sorting. As a result, we obtain a new ranking loss function which is an arbitrarily accurate approximation to the evaluation metric, thus closing the gap between the training and the evaluation of LTR models. We introduce two variants of the proposed loss function. Finally, the empirical evaluation shows that our proposed method outperforms ApproxNDCG, another differentiable approximation to NDCG, and is competitive with the state-of-the-art methods.

CCS CONCEPTS

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

KEYWORDS

Learning to Rank, ranking metric optimisaton, NDCG approximation

ACM Reference Format:
Przemysław Pobrotyn and Radosław Białobrzeski. 2021. NeuralNDCG: Direct Optimisation of a Ranking Metric via Differentiable Relaxation of Sorting. In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR eCom'21).* ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

Ranking is the problem of optimising, conditioned on some context, the ordering of a set of items in order to maximise a given metric. The metric is usually an Information Retrieval (IR) criterion chosen to correlate with user satisfaction. Learning to Rank (LTR) is a machine learning approach to ranking, concerned with learning the function which optimises the items' order from supervised data. In this work, without loss of generality, we assume our set of items are search results and the context in which we want to optimise their order is the user query.

Essentially, one would like to learn a function from search results into permutations. Since the space of all permutations grows factorially in the size of the search results set, the task of learning such a function directly becomes intractable. Thus, most common LTR algorithms resort to the approach known as *score & sort.* That is, instead of directly learning the correct permutation of the search results, one learns a *scoring function* which predicts relevancies of individual items, in the form of real-valued scores. Items are then sorted in the descending order of the scores and thus produced ordering is evaluated using an IR metric of choice. Typically, scoring functions are implemented as either gradient boosted trees [14] or Multilayer Perceptrons (MLP) [25]. Recently, there has been work in using the Transformer [28] architecture as a scoring function [21]. In order to learn a good scoring function, one needs a tagged dataset of query-search results pairs together with ground truth relevancy of each search result (in the context of a given query), as well as a loss function. There has been extensive research into constructing appropriate loss functions for LTR (see [19] for an overview of the field). Such loss functions fall into one of three categories: *pointwise*, *pairwise* or *listwise*. Pointwise approaches treat the problem as a simple regression or classification of the ground truth relevancy for each individual search result, foregoing possible interactions between items. In pairwise approaches, pairs of items are considered as independent variables and the function is learned to correctly indicate the preference among the pair. Examples include RankNet [5], LambdaRank [6] or LambdaMART [7]. However, IR metrics consider entire search results lists at once, unlike pointwise and pairwise algorithms. This mismatch has motivated listwise approaches, which compute the loss based on the scores of the entire list of search results. Two popular listwise approaches are ListNet [8] and ListMLE [29].

What these loss functions have in common is that they are either not connected or only loosely connected to the IR metrics used in the evaluation. The performance of LTR models is usually assessed using Normalised Discounted Cumulative Gain (NDCG) [16] or Mean Average Precision (MAP) [1]. Since such metrics rely on sorting the ground truth labels according to the scores predicted by the scoring function, they are either not differentiable or flat everywhere and thus cannot be used for gradient-based optimisation of the scoring function. As a result, there is a mismatch between

[*]Both authors contributed equally to this research.
[†]Corresponding author.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGIR eCom'21, July 15, 2021, Virtual Event, Montreal, Canada
© 2021 Copyright held by the owner/author(s).

objectives optimised by the aforementioned pairwise or listwise losses and metrics used for the evaluation, even though it can be shown that some of such losses provide upper bounds of IR measures [31], [30]. On the other hand, as demonstrated in [23], under certain assumptions on the class of the scoring functions, direct optimisation of IR measures on a large training set is guaranteed to achieve high test performance on the same IR measure. Thus, attempts to bridge the gap between LTR optimisation objectives and discontinuous evaluation metrics are an important research direction.

In this work, we propose a novel approach to directly optimise NDCG by approximating the sorting operator with NeuralSort [15]. Since the sorting operator is the source of discontinuity in NDCG (and other IR metrics), by substituting it with a differentiable approximation we obtain a smooth variant of the metric.

The main contributions of the paper are:

- We introduce NeuralNDCG, a novel smooth approximation of NDCG based on differentiable relaxation of the sorting operator. The variants of the proposed loss are discussed in Section 4.
- We evaluate a Context-Aware Ranker [21] trained with NeuralNDCG loss on Web30K [22] and Istella [12] datasets. We demonstrate favourable performance of NeuralNDCG as compared to baselines. In particular, NeuralNDCG outperforms ApproxNDCG [23], a competing method for direct optimisation of NDCG.
- We provide an open-source Pytorch [20] implementation allowing for the reproduction of our results available as part of the open-source allRank framework[1].

The rest of the paper is organised as follows. In Section 2, we review the related literature. In Section 3 we formalise the problem of LTR. In Section 4, we recap NeuralSort and demonstrate how it can be used to construct a novel loss function, NeuralNDCG. In Section 5 we discuss our experimental setup and results. In the final Section 6 we summarise our findings and discuss the possible future work.

2 RELATED WORK

As already mentioned in the introduction, most LTR approaches can be classified into one of three categories: pointwise, pairwise or listwise. For a comprehensive overview of the field and most common approaches, we refer the reader to [19].

In this work, we are concerned with the direct optimisation of non-smooth IR measures. Methods for optimisation of such metrics can be broadly grouped into two categories. The methods in the first category try to optimise the upper bounds of IR metrics as surrogate loss functions. Examples include SVM$^{\text{map}}$ [31] and SVM$^{\text{NDCG}}$ [9] which optimise upper bounds on $1 - \text{MAP}$ and $1 - \text{NDCG}$, respectively. On the other hand, ListNet was originally designed to minimise cross-entropy between predicted and ground truth top-one probability distributions, and as such its relation to NDCG was ill-understood. Only recently was it shown to bound NDCG and Mean Reciprocal Rank (MRR) for binary labels [3]. Further, a modification to ListNet was proposed in [2] for which it can be shown that it bounds NDCG also for the graded relevance labels. Popular methods like LambdaRank and LambdaMART forgo explicit formulation of the loss function and instead heuristically formulate the gradients based on NDCG considerations. Since the exact loss function is unknown, its theoretical relation to NDCG is difficult to analyse. The second category of methods aims to approximate an IR measure with a smooth function and directly optimise resulting surrogate function. Our method falls into this category. We propose to smooth-out NDCG by approximating non-continuous sorting operator used in the computation of that measure. Recent works proposing continuous approximation to sorting are already mentioned NeuralSort, SoDeep [13] and smooth sorting as an Optimal Transport problem [11]. We use NeuralSort for its firm mathematical foundation, the possibility to control the degree of approximation and ability to generalise beyond the maximum list length seen in training. SoDeep uses a deep neural network (DNN) and synthetic data to learn to approximate the sorting operator and as such lacks the aforementioned properties. Smooth sorting as Optimal Transport reports similar performance to NeuralSort at benchmark tasks and we aim to explore the use of it in NeuralNDCG in the future. By replacing the sorting operator with its continuous approximation, we obtain NeuralNDCG, a differentiable approximation of the IR measure. Other notable methods for direct optimisation of NDCG include:

- ApproxNDCG in which authors reformulated NDCG formula to involve summation over documents, not their ranks. As a result, they introduce a non-differentiable position function, which they approximate using a sigmoid. This loss has been recently revisited in a DNN setting in [4].
- SoftRank [27], where authors propose to smooth scores returned by the scoring function with equal variance Gaussian distributions: thus deterministic scores become means of Gaussian score distributions. Subsequently, they derive an $O(n^3)$ algorithm to compute Rank-Binomial rank distributions using the smooth scores. Finally, NDCG is approximated by taking its expectation w.r.t. the rank distribution.

3 PRELIMINARIES

In this section, we formalise the problem and introduce the notation used throughout the paper. Let $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X}^n \times \mathbb{Z}^n_{\geq 0}$ be a training example consisting of a vector $\boldsymbol{x}$ of n items x_i, $1 \leq i \leq n$, and a vector $\boldsymbol{y}$ of corresponding non-negative integer relevance labels. Note that each item x_i is itself a d-dimensional vector of numerical features, and should be thought of as representing a query-document pair. The set $\mathcal{X}$ is the space of all such vectors $\boldsymbol{x_i}$. Thus, a pair $(\boldsymbol{x}, \boldsymbol{y})$ represents a list of vectorised search results for a given query together with the corresponding ground truth relevancies. The dataset of all such pairs is denoted Ψ. The goal of LTR is to find a scoring function $f : \mathcal{X}^n \to \mathbb{R}^n$ that maximises the chosen IR metric on Ψ. The scoring function is learned by minimising the empirical risk $\mathcal{L}(f) = \frac{1}{|\Psi|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \Psi} \ell(\boldsymbol{y}, \boldsymbol{s})$ where $\ell(\cdot)$ is a loss function and $\boldsymbol{s} = f(\boldsymbol{x})$ is the vector of predicted scores. As discussed earlier, in most LTR approaches there is a mismatch between the loss function ℓ and the evaluation metric, causing a discrepancy between the learning procedure and its assessment. In this work, we focus on NDCG as our metric of choice and propose a new loss called NeuralNDCG, which bridges the gap between the training and the

[1]https://github.com/allegro/allRank

evaluation. Before we introduce NeuralNDCG, recall the definition of NDCG.

Definition 3.1. Let $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X}^n \times \mathbb{Z}^n_{\geq 0}$ be a training example and assume the documents in $\boldsymbol{x}$ have been ranked in the descending order of the scores computed using some scoring function f. Let r_j denote the relevance of the document ranked at j-th position, $g(\cdot)$ denote a gain function and $d(\cdot)$ denote a discount function. Then, the Discounted Cumulative Gain at k-th position ($k \leq n$) is defined as

$$\mathrm{DCG}@k = \sum_{j=1}^{k} g(r_j) d(j) \tag{1}$$

and Normalised Discounted Cumulative Gain at k is defined as

$$\mathrm{NDCG}@k = \frac{1}{\mathrm{maxDCG}@k} \mathrm{DCG}@k \tag{2}$$

where maxDCG@k is the maximum possible value of DCG@k, computed by ordering the documents in $\boldsymbol{x}$ by their decreasing ground truth relevancy.

Note that, typically, the discount function $d(\cdot)$ and the gain function $g(\cdot)$ are given by $d(j) = \frac{1}{\log_2(j+1)}$ and $g(r_j) = 2^{r_j} - 1$, respectively.

4 LOSS FORMULATION

In this section we define NeuralNDCG, a novel differentiable approximation to NDCG. It relies on NeuralSort, a smooth relaxation of the sorting operator. We begin by recalling NeuralSort, proceed to define NeuralNDCG and discuss its possible variants.

4.1 Sorting relaxation

Recall that sorting a list of scores $\boldsymbol{s}$ is equivalent to left-multiplying a column vector of scores by the permutation matrix $P_{\mathtt{sort}(s)}$ induced by permutation $\mathtt{sort}(\boldsymbol{s})$ sorting the scores. Thus, in order to approximate the sorting operator, it is enough to approximate the induced permutation matrix. In [15], the permutation matrix is approximated via a unimodal row stochastic matrix $\widehat{P}_{\mathtt{sort}(s)}(\tau)$ given by:

$$\widehat{P}_{\mathtt{sort}(s)}[i,:](\tau) = \mathrm{softmax}[((n+1-2i)\boldsymbol{s} - A_{\boldsymbol{s}}\mathbb{1})/\tau] \tag{3}$$

where $A_{\boldsymbol{s}}$ is the matrix of absolute pairwise differences of elements of $\boldsymbol{s}$ such that $A_{\boldsymbol{s}}[i,j] = |s_i - s_j|$, $\mathbb{1}$ denotes the column vector of all ones and $\tau > 0$ is a temperature parameter controlling the accuracy of approximation. For brevity, for the remainder of the paper we refer to $\widehat{P}_{\mathtt{sort}(s)}(\tau)$ simply as $\widehat{P}$.

Note that the temperature parameter τ allows to control the trade-off between the accuracy of the approximation and the variance of the gradients. Generally speaking, the lower the temperature, the better the approximation at the cost of a larger variance in the gradients. In fact, it is not difficult to demonstrate that:

$$\lim_{\tau \to 0} \widehat{P}_{\mathtt{sort}(s)}(\tau) = P_{\mathtt{sort}(s)} \tag{4}$$

(see [15] for proof). This fact will come in handy once we define NeuralNDCG.

An approximation of a permutation matrix by Equation 3 is a deterministic function of the predicted scores. Authors of NeuralSort proposed also a stochastic version, by deriving a reparametrised sampler for a Plackett-Luce family of distributions. Essentially, they propose to perturb scores s with a vector g of i.i.d. Gumbel perturbations with zero mean and a fixed scale β to obtain perturbed scores $\tilde{s} = \beta \log s + g$. Perturbed scores are then used in place of deterministic scores in the formula for $\widehat{P}$.

We experimented with both deterministic and stochastic approximations to sorting and found them to yield similar results. Thus, for brevity, in this work we focus on the deterministic variant.

4.2 NeuralNDCG

If the ground truth permutation is known, one could minimise the cross-entropy loss between the ground truth permutation matrix and its approximation given by $\widehat{P}$, as done in the experiments section in [15]. However, for many applications, including ranking, the exact ground truth permutation is not known. Relevance labels of individual items produce many possible valid ground truth permutation matrices. Thus, instead of optimising the cross-entropy loss, we use NeuralSort to introduce NeuralNDCG, a novel loss function appropriate for LTR.

Given a list of documents $\boldsymbol{x}$, its corresponding vector of scores $\boldsymbol{s} = f(\boldsymbol{x})$ and the ground truth labels $\boldsymbol{y}$ we first find the approximate permutation matrix $\widehat{P}$ induced by the scores $\boldsymbol{s}$ using Equation 3. We then apply the gain function g to the vector of ground truths $\boldsymbol{y}$ and obtain a vector $g(\boldsymbol{y})$ of gains per document. We then left-multiply the column vector $g(\boldsymbol{y})$ of gains by $\widehat{P}$ and obtain an "approximately" sorted version of the gains, $\widehat{g(\boldsymbol{y})}$. Another way to think of that approximate sorting is that the k-th row of $\widehat{P}$ gives weights of documents x_i in the computation of gain at rank k after sorting. Gain at rank k is then the weighted sum of ground truth gains, weighted by the entries in the k-th row of $\widehat{P}$. Note that after the approximate sorting the actual integer values of ground truth gains become "distorted" and are not necessarily integers anymore (See Table 1 for example). In particular, the sum of quasi-sorted gains $\widehat{g(\boldsymbol{y})}$ may differ from that of the original vector $g(\boldsymbol{y})$. This leads to a peculiar behaviour of NeuralNDCG near the discontinuities of true NDCG (Figure 1), which may be potentially harmful for optimisation using Stochastic Gradient Descent [24]. Since $\widehat{P}$ is obtained by applying row-wise softmax operator, the permutation matrix $\widehat{P}$ is row-stochastic but not-necessarily column-stochastic (i.e. each column does not necessarily sum to one), an individual ground truth gain $g(\boldsymbol{y})_j$ may have corresponding weights in the rows of $\widehat{P}$ that do not sum to one (and, in particular, may sum to more than one), so it will overcontribute to the total sum of $\widehat{g(\boldsymbol{y})}$. To alleviate that problem, we additionally perform Sinkhorn scaling [26] on $\widehat{P}$ (i.e. we iteratively normalize all rows and columns until convergence[2]) before using it for quasi-sorting. This way, the columns also sum to one and the approximate sorting is smoothed-out (again, see Figure 1). The remaining steps are identical to the computation of NDCG@k, with the exception that the gain of the relevance function r_j is replaced with the j-th coordinate of quasi-sorted gains $\widehat{g(\boldsymbol{y})}$. For the discount function d, we use the usual inverse logarithmic discount and for the gain function g we used the usual power function. For the computation of the maxDCG, we

[2]We stop the procedure after 30 iterations or when the maximum difference between row or column sum and one is less than 10^{-6}, whatever happens first.

Table 1: Approximate sorting with NeuralSort. Given ground truth $\boldsymbol{y} = [4, 2, 1, 0, 4, 3]$ and predicted scores $s = [0.5, 0.2, 0.1, 0.01, 0.65, 0.3]$, $\boldsymbol{y}$ is sorted by $\widehat{P}$ for different values of τ. Exact sorting is shown in the first row.

	Quasi-sorted ground truth						**Sum after sorting**
$\lim_{\tau\to 0}$	4	4	3	2	1	0	14
$\tau = 0.01$	4	4	3	2	0.99992	0.00012339	14.00004339
$\tau = 0.1$	3.9995	3.8909	2.8239	1.9730	0.9989	0.3136	13.9998
$\tau = 1$	3.3893	2.9820	2.4965	2.0191	1.6097	1.2815	10.388

use original ground truth labels $\boldsymbol{y}$. To find NeuralNDCG at rank k, we simply truncate quasi-sorted gains to the k-th position and compute the maxDCG at k-th rank.

We thus obtain the following formula for NeuralNDCG:

$$\text{NeuralNDCG}_k(\tau)(\boldsymbol{s}, \boldsymbol{y}) = N_k^{-1} \sum_{j=1}^{k} (\text{scale}(\widehat{P}) \cdot g(\boldsymbol{y}))_j \cdot d(j) \qquad (5)$$

where N_k^{-1} is the maxDCG at k-th rank, scale$(\cdot)$ is Sinkhorn scaling and $g(\cdot)$ and $d(\cdot)$ are their gain and discount functions. Note that the summation is over the first k ranks.

Finally, since the popular autograd libraries provide means to minimise a given loss functions, we use $(-1) \times$ NeuralNDCG for optimisation.

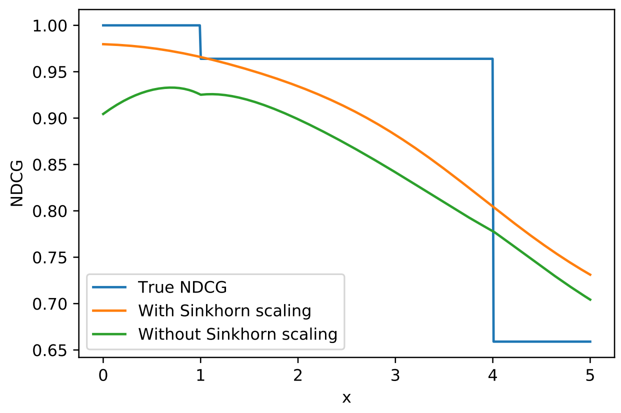

Figure 1: Given ground truth $y = [2, 1, 0, 0, 0]$ and a list of scores $s = [4, 1, 0, 0, x]$, we vary the value of the score x and plot resulting NDCG induced by the scores along with NeuralNDCG ($\tau = 1.0$) with and without Sinkhorn scaling of $\widehat{P}$.

4.3 NeuralNDCG Transposed

In the above formulation of NeuralNDCG, the summation is done over the ranks and gain at each rank is computed as a weighted sum of all gains, with weights given by the rows of $\widehat{P}$. We now provide an alternative formulation of NeuralNDCG, called NeuralNDCG Transposed (NeuralNDCG^T for short), where the summation is done over the documents, not their ranks.

As previously, let $\boldsymbol{x}$ be a list of documents with corresponding scores $\boldsymbol{s}$ and ground truth relevancies $\boldsymbol{y}$. We begin by finding the approximate permutation matrix $\widehat{P}$. Since we want to sum over the documents and not their ranks, we need to find the weighted average of discounts per document, not the weighted average of gains per rank as before. To this end, we transpose $\widehat{P}$ to obtain an approximation $\widehat{P}^T$ of the inverse of the permutation matrix corresponding to sorting the documents $\boldsymbol{x}$ by their corresponding scores $\boldsymbol{y}$. Thus, $\widehat{P}^T$ can be thought of as an approximate *unsorting* matrix - when applied to sorted documents (ranks), it will (approximately) recover their original ordering. Since $\widehat{P}$ is row-stochastic, $\widehat{P}^T$ is column-stochastic. As we want to apply it by left-multiplication, we want it to be row-stochastic. Thus, similarly to before, we perform Sinkhorn scaling of $\widehat{P}^T$. After Sinkhorn scaling, the k-th row of $\widehat{P}^T$ can be thought of as giving the weights of different ranks when computing the discount of the k-th document. We can now find the vector of the weighted averages of discounts per document by computing $\widehat{P}^T\boldsymbol{d}$, where $\boldsymbol{d}$ is the vector of logarithmic discounts per rank ($\boldsymbol{d}_j = d(j)$). Note that since we want to perform summation over the documents, not ranks, it is not enough to sum the first k elements to truncate NDCG to the k-th position. Instead, the entries of the discounts vector $\boldsymbol{d}$ corresponding to ranks $j > k$ are set to 0. This way, the documents which would end up at ranks $j > k$ after sorting end up having weighted discounts being close to 0, and equal to 0 in the limit of the temperature τ. Thus, even though the summation is done over all documents, we still recover NDCG@k.

Hence, NeuralNDCG^T is given by the following formula:

$$\text{NeuralNDCG}^T{}_k(\tau)(\boldsymbol{s}, \boldsymbol{y}) = N_k^{-1} \sum_{i=1}^{n} g(\boldsymbol{y}_i) \cdot (\text{scale}(\widehat{P}^T) \cdot \boldsymbol{d})_i \qquad (6)$$

where N_k^{-1} is the maxDCG at k-th rank, scale$(\cdot)$ is Sinkhorn scaling, $g(\cdot)$ is the gain function, $\boldsymbol{d}$ is the vector of logarithmic discounts per rank set to 0 for ranks $j > k$, and the summation is done over all n documents.

4.4 Properties of NeuralNDCG

By Equation 4, in the limit of the temperature, the approximate permutation matrix $\widehat{P}$ becomes the true permutation matrix P. Thus, as the temperature approaches zero, NeuralNDCG approaches true NDCG in both its variants. See Figure 2 for examples of the effect of the temperature on the accuracy of the approximation.

Comparing to ApproxNDCG, our proposed approximation to NDCG showcases more favourable properties. We can easily compute NDCG at any rank position k, whereas in ApproxNDCG, one needs to further approximate the truncation function using an

Table 2: Dataset statistics

Dataset	Features	Queries in training	Queries in test	Empty queries
Web30K	136	18919	6306	982
Istella	220	23219	9799	50

approximation of the position function. This approximation of an approximation leads to a compounding of errors. We deal away with that problem by using a single approximation of the permutation matrix. Furthermore, the approximation of the position function in ApproxNDCG is done using a sigmoid function, which may lead to the vanishing gradient problem.

SoftRank suffers from a high computational complexity of $O(n^3)$: in order to compute all the derivatives required by the algorithm, a recursive computation is necessary. Authors relieve that cost by approximating all but a few of the Rank-Binomial distributions used, but at a cost of the accuracy of their solution. On the other hand, computation of $\widehat{P}$ is of $O(n^2)$ complexity.

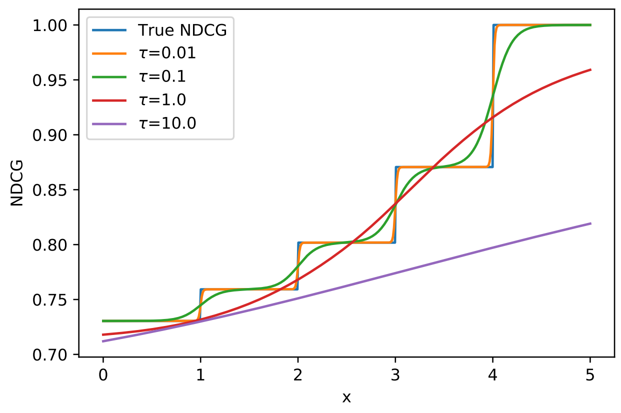

Figure 2: Given ground truth $y = [1, 2, 3, 4, 5]$ and a list of scores $s = [1, 2, 3, 4, x]$, we vary the value of the score x and plot resulting NDCG induced by the scores along with NeuralNDCG for different temperatures τ.

5 EXPERIMENTS

This section describes the experimental setup used to empirically verify the proposed loss functions.

5.1 Datasets

Experiments were conducted on two datasets: Web30K and Istella[3]. Both datasets consists of queries and associated search results. Each query-document pair is represented by a real-valued feature vector and has an associated graded relevance on the scale from 0 (irrelevant) to 4 (highly relevant). For both datasets, we standardise the features, log-transforming selected ones, before feeding them to the learning algorithm. Since the lengths of search results lists in the datasets are unequal, we padded or subsampled to equal length for training, but used the full list length for evaluation. Web30K comes split into five folds. However, following the common practice in the field, we report results obtained on Fold 1 of the data. We used 60% of the data for training, 20% for validation and hyperparameter tuning and the remaining 20% for testing. Istella datasets come partitioned into a training and a test fold according to a 80%-20% schema. We additionally split the training data into training and validation data to obtain a 60%/20%/20% split, similarly to Web30K. We tune the hyperparameters of our models on the validation data and report performance on the test set, having trained the best models on the full training fold. In both datasets there are a number of queries for which the associated search results list contains no relevant documents (i.e. all documents have label 0). We refer to these queries as *empty queries*. For such queries, the NDCG of their list of results can be arbitrarily set to either 0 or 1. To allow for a fair comparison with the current state of the art, we followed default XGBoost [10] and LightGBM [17] implementation of setting NDCG of such lists to 1 during the evaluation. Table 2 summarizes the characteristics of the datasets used.

5.2 Scoring function

For the scoring function f, we used the Context-Aware Ranker, a ranking model based on the Transformer architecture. The model can be thought of as the encoder part of the Transformer, taking raw features of items present in the same list as input and outputting a real-valued score for each item. Given the ubiquity of Transformer-based models in the literature, we refer the reader to [21] for the details of the architecture used. Compared to the original network described in [21], we used smaller architectures. For both datasets, we used an architecture consisting of 2 encoder blocks of a single attention head each, with a hidden dimension of 384. The dimensionality of initial fully-connected layer was set to 96 for models trained on Web30K and 128 for models trained on Istella. We did not apply any activation on the output except for NeuralNDCG and NeuralNDCG^T. It exhibited suboptimal performance without any nonlinear output activation function and, in this case, we applied Tanh to the output. For both datasets, the same architectures were used across all loss functions.

5.3 Training hyperparameters

In all cases, we used Adam [18] optimiser and set the learning rate to 0.001. The batch size was set to 64 (Web30K) or 110 (Istella) and search results lists were truncated or padded to the length of 240 when training. We trained the networks for 100 epochs, decaying the learning rate by the factor of 0.1 after 50 epochs.

[3]There are a few variants of this dataset, we used the Istella full dataset.

Table 3: Test NDCG on Web30K and Istella. Boldface is the best performing loss column-wise.

Loss	WEB30K		Istella	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
NeuralNDCG@5	50.32	52.01	65.32	69.97
NeuralNDCG@10	50.89	52.77	65.65	70.68
NeuralNDCG@max	**51.56**	53.46	65.69	70.55
NeuralNDCG^T@5	50.50	52.14	65.46	69.95
NeuralNDCG^T@10	50.85	52.70	**66.02**	71.02
NeuralNDCG^T@max	51.45	**53.49**	65.60	70.53
ApproxNDCG	49.07	50.90	63.14	67.94
ListNet	50.75	52.80	65.62	70.70
ListMLE	49.81	51.82	59.85	66.24
RankNet@5	49.14	50.75	64.45	68.74
RankNet@10	50.95	52.69	65.75	70.68
RankNet@max	49.84	51.82	64.57	70.37
LambdaRank@5	48.70	50.10	63.50	67.75
LambdaRank@10	49.66	51.34	65.21	69.82
LambdaRank@max	51.55	53.47	65.90	**71.09**
RMSE	50.51	52.46	65.62	70.76
XGBoost	46.80	49.17	61.04	65.74

5.4 Loss functions

We compared the performance of variants of NeuralNDCG against the following loss functions. For a pointwise baseline, we used a simple RMSE of predicted relevancy. Specifically, the output of the network f is passed through a sigmoid function and then multiplied by the number of relevance levels. The root mean squared difference of this score and the ground truth relevance is the loss. Pairwise losses we compared with consist of RankNet and LambdaRank. Similarly to NeuralNDCG, these losses support training with a specific rank cutoff. We thus train models with these losses at ranks 5, 10 and at the maximum rank. The two most popular listwise losses are ListNet and ListMLE, and we, too, included them in our evaluation. Finally, the other method of direct optimisation of NDCG which we compared with was ApproxNDCG. We did not compare with SoftRank, as its $O(n^3)$ complexity proved prohibitive. We tuned ApproxNDCG and NeuralNDCG smoothness hyperparameters for optimal performance on the test set. Both ApproxNDCG's α and NeuralNDCG's τ parameter were set to 1 as other values in the $[0.01; 100]$ interval did not show any improvement.

5.5 Results

For both datasets, we report models performance in terms of NDCG@5 and NDCG@10. Results are collected in Table 3. Both NeuralNDCG variants in every rank cutoff setting outperform ApproxNDCG on both datasets in all metrics reported. Moreover, NeuralNDCG variants with specific rank cutoffs provide the best performance among all losses in both metrics on the WEB30K dataset and NDCG@5 on the Istella dataset. For reference, we also report the results of a GBDT model trained with XGBoost [10] and objective rank:pairwise (as rank:ndcg is known to yield suboptimal results[4]).

[4]For details, please visit https://github.com/dmlc/xgboost/issues/6352.

6 CONCLUSIONS

In this work we introduced NeuralNDCG, a novel differentiable approximation of NDCG. By substituting the discontinuous sorting operator with NeuralSort, we obtain a robust, efficient and arbitrarily accurate approximation to NDCG. Not only does it enjoy favourable theoretical properties, but also proves to be effective in empirical evaluation, yielding competitive performance, on par with LambdaRank. This work can easily be extended to other rank-based metrics like MAP; a possibility we aim to explore in the future. Another interesting extension of this work would be the substitution of NeuralSort with another method of approximation of the sorting operator, most notably the method treating sorting as an Optimal Transport problem [11].

REFERENCES

[1] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] Sebastian Bruch. 2019. An Alternative Cross Entropy Loss for Learning-to-Rank. arXiv:1911.09798 [cs.LG]

[3] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval* (Santa Clara, CA, USA) *(ICTIR '19)*. Association for Computing Machinery, New York, NY, USA, 75–78. https://doi.org/10.1145/3341981.3344221

[4] Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and 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* (Paris, France) *(SIGIR'19)*. Association for Computing Machinery, New York, NY, USA, 1241–1244. https://doi.org/10.1145/3331184.3331347

[5] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to Rank Using Gradient Descent. In *Proceedings of the 22Nd International Conference on Machine Learning* (Bonn, Germany) *(ICML '05)*. ACM, New York, NY, USA, 89–96. https://doi.org/10.1145/1102351.1102363

[6] Christopher J. Burges, Robert Ragno, and Quoc V. Le. 2007. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman (Eds.). MIT Press, 193–200. https://papers.nips.cc/paper/2971-learning-to-rank-with-nonsmooth-cost-functions

[7] Christopher J. C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report. Microsoft Research. http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf

[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 *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, Oregon, USA) *(ICML '07)*. ACM, New York, NY, USA, 129–136. https://doi.org/10.1145/1273496.1273513

[9] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. 2008. Structured Learning for Non-Smooth Ranking Losses. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Las Vegas, Nevada, USA) *(KDD '08)*. Association for Computing Machinery, New York, NY, USA, 88–96. https://doi.org/10.1145/1401890.1401906

[10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[11] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. 2019. Differentiable Ranking and Sorting using Optimal Transport. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 6858–6868. http://papers.nips.cc/paper/8910-differentiable-ranking-and-sorting-using-optimal-transport.pdf

[12] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees. *ACM Trans. Inf. Syst.* 35, 2, Article 15 (Dec. 2016), 31 pages. https://doi.org/10.1145/2987380

[13] Martin Engilberge, Louis Chevallier, Patrick Perez, and Matthieu Cord. 2019. SoDeep: A Sorting Deep Net to Learn Ranking Loss Surrogates. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[14] Jerome H. Friedman. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2000), 1189–1232.

[15] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic Optimization of Sorting Networks via Continuous Relaxations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1eSS3CcKX

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446. https://doi.org/10.1145/582415.582418

[17] 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*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 3146–3154. http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[18] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[19] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (March 2009), 225–331. https://doi.org/10.1561/1500000016

[20] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[21] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. 2020. Context-Aware Learning to Rank with Self-Attention. In *SIGIR eCom '20*. Virtual Event, China.

[22] Tao Qin and T. M. Liu. 2013. Introducing LETOR 4.0 Datasets. *ArXiv* abs/1306.2597 (2013).

[23] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Inf. Retr.* 13 (08 2010), 375–397. https://doi.org/10.1007/s10791-009-9124-x

[24] H. Robbins and S. Monro. 1951. A stochastic approximation method. *Annals of Mathematical Statistics* 22 (1951), 400–407.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation.* Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[26] Richard Sinkhorn. 1964. A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices. *Ann. Math. Statist.* 35, 2 (06 1964), 876–879. https://doi.org/10.1214/aoms/1177703591

[27] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. SoftRank: optimizing non-smooth rank metrics. 77–86. https://doi.org/10.1145/1341531.1341544

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[29] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th International Conference on Machine Learning* (Helsinki, Finland) *(ICML '08)*. ACM, New York, NY, USA, 1192–1199. https://doi.org/10.1145/1390156.1390306

[30] Jun Xu and Hang Li. 2007. AdaRank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual ACM SIGIR conference.* ACM, Amsterdam, The Netherlands, 391–398. https://doi.org/10.1145/1277741.1277809

[31] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. 2007. A Support Vector Method for Optimizing Average Precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Amsterdam, The Netherlands) *(SIGIR '07)*. Association for Computing Machinery, New York, NY, USA, 271–278. https://doi.org/10.1145/1277741.1277790