LambdaRank Gradients are Incoherent

Federico Marcuzzi
federico.marcuzzi@unive.it
Università Ca' Foscari Venezia
Venice, Italy

Claudio Lucchese
claudio.lucchese@unive.it
Università Ca' Foscari Venezia
Venice, Italy

Salvatore Orlando
orlando@unive.it
Università Ca' Foscari Venezia
Venice, Italy

ABSTRACT

In Information Retrieval (IR), the Learning-to-Rank (LTR) task requires building a ranking model that optimises a specific IR metric. One of the most effective approaches to do so is the well-known LambdaRank algorithm. LambdaRank uses gradient descent optimisation, and at its core, it defines approximate gradients, the so-called *lambdas*, for a non-differentiable IR metric. Intuitively, each lambda describes how much a document's score should be "pushed" up/down to reduce the ranking error.

In this work, we show that lambdas may be incoherent w.r.t. the metric being optimised: e.g., a document with high relevance in the ground truth may receive a smaller gradient push than a document with lower relevance. This behaviour goes far beyond the expected degree of approximation. We analyse such behaviour of LambdaRank gradients and we introduce some strategies to reduce their incoherencies. We demonstrate through extensive experiments, conducted using publicly available datasets, that the proposed approach reduces the frequency of the incoherencies in LambdaRank and derivatives, and leads to models that achieve statistically significant improvements in the NDCG metric, without compromising the training efficiency.

CCS CONCEPTS

• **Information systems → Learning to rank**; **Retrieval effectiveness**.

KEYWORDS

Information Retrieval; Learning to Rank; LambdaRank

ACM Reference Format:
Federico Marcuzzi, Claudio Lucchese, and Salvatore Orlando. 2023. LambdaRank Gradients are Incoherent. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614948

1 INTRODUCTION

Information Retrieval (IR) is a research field that focuses on finding relevant items (usually text documents) within large collections of an unstructured and disordered nature that satisfies an information need. Learning to Rank (LTR), namely machine learning applied to the task of document ranking, is nowadays one of the most popular techniques adopted in modern IR. LTR includes supervised learning algorithms to construct rankers that aim to sort by user relevance a huge set of items, which mostly contain non-relevant items and only a small fraction of relevant ones [20, 26] A significant difficulty in learning a ranker is that typically IR metrics depend on the sorted list of documents. Therefore, an objective function that makes use of IR metrics cannot be directly optimised by gradient descent-based methods since metrics are either non-differentiable or flat everywhere with respect to the model parameters [1, 3, 12].

Despite the non-differentiability of IR metrics, many LTR algorithms are gradient-based, and either optimise an approximate version of the ranking metric or build gradients based on heuristic approximations such as LambdaRank [3]. LambdaRank avoids the definition of an approximate loss function; rather, it heuristically defines gradients that specify whether a document score should be increased or decreased to improve the ranking quality. LambdaRank is an algorithm originally designed for artificial neural networks that has been considered state-of-the-art in LTR until it was supplanted by LambdaMART [2], its analogous version based on gradient-boosted decision trees. Since both LambdaRank and LambdaMART are based on heuristics, their gradients are not an exact computation of the derivative of an IR metric.

In this work, we show that LambdaRank heuristics (LambdaLoss Framework [26], LambdaMART, etc.) have an inherent flaw and they can generate incoherent gradients. Later in Fig. 1 we show a few examples where the most relevant document in the result list does not get the largest gradient and therefore it is impossible for the learned model to rank it in the top position. We call *gradient incoherency* such phenomena where a relevant document receives a smaller gradient than a less relevant one. We are aware that gradients are approximate and therefore trade-offs need to be made in order to optimise non-differentiable functions. However, such incoherency may undermine the learning process. Moreover, this phenomenon is more apparent with the use of truncated metrics optimisation where we would like the model to focus on the top positions, but the gradients are unable to push upwards the most relevant documents.

The contributions of this work are as follows. *i)* We bring to light the issue of gradient incoherencies affecting LambdaRank, which has not been previously shown in the literature. *ii)* We show how truncated metric optimisation exacerbates the phenomenon of gradient incoherencies and undermines the aim of truncation, which is to ensure that the user encounters the most relevant documents among the first positions. *iii)* We propose an improvement over the LambdaRank gradient computation to optimise truncated ranking metrics. Specifically, we propose Lambda-*eX*, which *extends* the set of document pairs considered by LambdaRank when computing gradients.

We validate experimentally our results on five publicly available datasets. We show how Lambda-*eX* can reduce the number of

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0124-5/23/10.
https://doi.org/10.1145/3583780.3614948

queries affected by gradient incoherencies introduced by truncated metric optimisation. This reduction is significantly large during the early stage of the training, which allows Lambda-*eX* to achieve high-quality performance after a few trees. Finally, we show that optimising truncated metrics can accelerate training time due to a lower number of partial derivatives to be computed and that Lambda-*eX* can achieve statistically significant improvements while maintaining the same train efficiency as truncated LambdaRank.

2 BACKGROUND

One of the most common IR applications of LTR is to make use of machine learning to implement the re-ranking stage of the query processing pipeline (i.e., first-stage retrieval and then re-ranking). The purpose of LTR is to train a ranker f that, given a query q and a set of candidate documents $D = \{d_1, \ldots, d_n\}$, returns a ranking π over the set of documents, according to which documents are sorted and presented to the user. To generate the ranking π, the ranker predicts a score $s_i = f(d_i)$, for each $d_i \in D$, and then sorts the documents in descending score order. As a result, $\pi[i]$ denotes the position of document d_i in the ranking; therefore, we have $\pi[i] = 1$ when d_i appears at the top-1 position in the ranked list.

During the training of a LTR model, the ranker takes as input a set of triples (q, D, Y) called training set, where $q \in Q$ is a query, D is a set of candidate documents for q among the available document collection $\mathcal{D}$, and Y includes the true relevance labels y_i for each $d_i \in D$. Typically, relevance labels are integers in $\mathcal{Y} = \{0, 1, 2, 3, 4\}$, where 0 stands for non-relevant judgement and 4 is the maximum relevance to query q.

A learning algorithm aims at finding the ranker that optimises some given ranking metric. One of the most well-known and widely used IR metrics for measuring the quality of a ranked list is *Normalized Discounted Cumulative Gain* (NDCG) [15]. NDCG is the normalised version of Discounted Cumulative Gain, which accumulates a gain for each ranking position. DCG is composed of two components: the gain $\mathrm{G}_i = 2^{y_i} - 1$ that document d_i, with relevance y_i, brings to the final ranking, and the discounting $\mathrm{D}_i = \log_2(1 + i)$ that discounts the document's gain according to its position i in the list. In fact, a very relevant document in the last position contributes much less to NDCG than in the first position.

Below is the definition of NDCG:

$$\mathrm{NDCG} = \frac{\mathrm{DCG}}{\mathrm{IDCG}} = \frac{1}{\mathrm{IDCG}} \sum_{i=1}^{n} \frac{\mathrm{G}_i}{\mathrm{D}_i} = \frac{1}{\mathrm{IDCG}} \sum_{i=1}^{n} \frac{2^{y_i} - 1}{\log_2(1 + \pi[i])},$$

where IDCG is the ideal DCG of the ground truth ranking. Normalisation prevents the model from favouring long queries since queries with a lot of candidate documents are likely to have a higher DCG. Several other ranking metrics can be similarly formulated with different gains G_i and discounts D_i and with proper normalisation.

2.1 Gradient-descent based learning and LambdaRank

Gradient-based learning algorithms, such as artificial neural networks or gradient-boosted decision trees, run iterative updates to build a ranker that minimises a given cost function C. For instance, artificial neural networks compute the gradient direction $\partial C/\partial w_j$ to update each network weight w_j at each batch processed. Similarly, gradient-boosted decision trees iteratively learn a new tree that approximates $\partial C/\partial s_i$ for each document d_i in the training set. In both cases, directly or indirectly, a key step is the computation of $\partial C/\partial s_i$. Unfortunately, most IR metrics are rank-based: they depend on π rather than on s_i. This makes the cost function either flat, i.e., modifications of s_i do not change π and therefore do not change the cost C, or non-differentiable, i.e., modifications of s_i change π causing a non-smooth change of the cost C.

Most approaches drive the learning process by means of a proxy cost function that is differentiable. One of the most relevant approaches is LambdaRank [3]. LambdaRank's cost function stems from the RankNet cost [4] which is further enhanced by considering the impact on the IR metric at hand. The gradient $\partial C/\partial s_i$ is computed on the basis of pair-wise *lambdas* λ_{ij}. Given a document pair d_i and d_j such that d_i is more relevant than d_j, i.e., $y_i > y_j$, we have that:

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} \left|\Delta \mathrm{Z}_{ij}\right|, \tag{1}$$

where $\sigma = 1$, and $|\Delta \mathrm{Z}_{ij}|$ is the amount of change in the IR metric Z generated by swapping the rank positions of d_i and d_j while leaving the rank positions of all other documents unchanged. The value of λ_{ij} estimates the change on the cost function C when the distance between the two scores s_i and s_j is modified. Note that if two documents have the same relevance label, then $\lambda_{ij} = 0$ due to the fact that $\Delta \mathrm{Z}_{ij} = 0$. We recall that λ_{ij} implements the derivative of the RankNet cost function multiplied by $|\Delta \mathrm{Z}_{ij}|$. The RankNet cost increases if the two documents are not in the correct order, and converges asymptotically to 0 if documents are in the correct order with a large gap in their scores. The $|\Delta \mathrm{Z}_{ij}|$ component boosts the error when this has a significant impact on the specific IR metric.

The gradient of the single document is finally computed as:

$$\lambda_i = \sum_{j:(i,j) \in I} \lambda_{ij} - \sum_{k:(k,i) \in I} \lambda_{ki}, \tag{2}$$

where I is the set of ordered pairs (i, j) such that $y_i > y_j$, i.e., $I = \{(i, j) \mid d_i, d_j \in D \wedge y_i > y_j\}$. In regard to the asymptotic complexity of computing λ_{ij}, Eq. 2 requires to evaluate $O(n^2)$ document pairs with n being the number of candidate documents in D for the given query.

According to [3], when maximising an IR metric such as NDCG, the lambdas are formulated as $\partial U/\partial s_i$ where U is the utility function (metric) being maximised, rather than a cost to be minimised. Moreover, the sign of the various λ_{ij} is set so that the most relevant document d_i receives a positive gradient update, while d_j receives a negative update and is *pushed* down through the ranks π.

Before going through the next section, let's focus on the $|\Delta \mathrm{Z}_{ij}|$ in Eq. 1. Indeed, Z could be any ranking metric such as NDCG, ERR [7], etc. In this work, we focus on NDCG, and therefore $|\Delta \mathrm{NDCG}_{ij}|$ is defined as the difference between the NDCG computed on the current ranking π, and the NDCG computed on the ranking that results from swapping the two documents at ranks $\pi[i]$ and $\pi[j]$. This can be computed efficiently so that we can define λ_{ij} as:

$$\lambda_{ij} = \frac{1}{1 + e^{(s_i - s_j)}} \frac{1}{\mathrm{IDCG}} \left|\mathrm{G}_i - \mathrm{G}_j\right| \left|\frac{1}{\mathrm{D}_i} - \frac{1}{\mathrm{D}_j}\right|. \tag{3}$$

The gradient λ_{ij} has, therefore, three components: the RankNet cost, the gain difference and the difference of the inverse discount.

2.2 Optimisation of truncated metrics

Real-world applications of information retrieval systems mostly try to optimise the effectiveness for the first k results only. This manner is strictly related to user behaviour [12]. When users scan a list of results, they focus more on the first k (i.e., 5/10) results and do not look at all the thousands of results in the list. IR metrics naturally provide a truncated version with a *cutoff threshold* k. For instance, NDCG@k is computed by considering only the contribution of the top-k ranked documents. Truncated metrics are the IR metrics of interest to evaluate the goodness of a ranker in most application scenarios. Therefore, according to the empirical risk minimisation principle, the optimisation of a truncated metric is expected to be more effective than its un-truncated variant.

The introduction of a truncated metric at training time also brings a straightforward efficiency improvement. In Eq. 2, the computation of all λ_i requires computing the pair-wise gradients λ_{ij} for every pair of documents $d_i, d_j \in D$. Under a truncated metric, documents ranked beyond k are not considered, and therefore a pair of such documents has a value of $|\Delta Z_{ij}@k|$ equal to 0. We can thus limit the pairs to be considered to those that contain at least one document in the top-k. To do so, the gradients λ_i in Eq. 2 are computed by replacing the set I with a I_τ defined as follows:

$$I_\tau = \{(i, j) \mid d_i, d_j \in D \wedge y_i > y_j \wedge \min(\pi[i], \pi[j]) \leq \tau\}. \quad (4)$$

Hereinafter we denote by k the *cutoff* threshold used by the evaluation metric and by τ the *truncation level* [12] optimised during the training. Note that k and τ do not need to be the same. When $\tau = k$ we are maximising the truncated metric with cutoff k, while when $\tau = +\infty$ we are maximising the un-truncated metric.

Last but not least, the introduction of τ reduces the asymptotic complexity of the objective function from $O(n^2)$ to $O(\tau n)$. A significant reduction of the training time is achieved when $\tau \ll n$.

Despite being a tiny detail, we remark that we assume the use of Eq. 1 (and Eq. 3) without modification even in the presence of a truncation level, i.e., ΔZ is used rather than $\Delta Z@k$, as this is common practice in the most popular implementations, e.g., LightGBM [17].

3 GRADIENT COHERENCY AND LAMBDARANK

LambdaRank and its variants provide a smooth approximation of commonly used IR metrics. Yet, we would like this approximation to provide some basic guarantees. We thus introduce a *coherency* property defined as follows.

Definition 3.1 (Gradient Coherency). Given two documents d_i and d_j such that $y_i > y_j$ and $\pi[i] > \pi[j]$, i.e., d_i is more relevant than d_j but it is ranked at a worse position, we say that the gradients of the utility function U are *coherent* at d_i, d_j if it holds that:

$$\frac{\partial U}{\partial s_i} \geq \frac{\partial U}{\partial s_j}.$$

The above definition states that if two documents d_i and d_j are misranked, we would like the computed gradient to be larger at the most relevant document d_i. This is because pushing up d_i more than d_j may restore the ideal ordering. Conversely, when the gradient coherency does not hold, pushing up d_j more than d_i may only worsen the current ranking. Despite its simplicity, the gradient coherency property is not easy to satisfy, and, indeed, it is not enjoyed by LambdaRank gradients.

3.1 Incoherencies in LambdaRank gradients

In Fig. 1(a) we show an example of LambdaRank gradients computation when maximising NDCG@1 with a truncation level $\tau = 1$. Suppose this is the ranking after a given number of iterations, epochs, or boosting rounds, of a gradient-based optimisation algorithm. We have that the most relevant document $d_\triangle$ is currently ranked second, while a less relevant document $d_\star$ is ranked first. Then we have three other non-relevant documents. The arrows depict the computed gradients and, as expected, relevant documents get an upward push, while non-relevant documents get a downward push. However, we have that document $d_\triangle$ gets a smaller gradient than $d_\star$, i.e., $\lambda_\triangle < \lambda_\star$, meaning that the most relevant document $d_\triangle$ will not be able to reach the top position in the next iteration. On the contrary, the gap between $d_\star$ and $d_\triangle$ is meant to increase in favour of the least relevant $d_\star$. In fact, this is a gradient incoherency according to Def. 3.1.

To clarify this behaviour, in Tab 1 we report the computation of the document gradients λ_i as a function of the pair-wise λ_{ij} according to Eq. 3. Recall, that λ_{ij} is considered *if and only if* $(i, j) \in I_\tau$, i.e., at least one of the two documents is ranked above the truncation level τ. Let's focus on the relevant documents. Document $d_\star$ receives a negative contribution $-\lambda_{\triangle\star}$ from the most relevant document $d_\triangle$, and three positive contributions from the non-relevant documents. Document $d_\triangle$ is below the truncation level, and therefore its gradient is simply $\lambda_\triangle = +\lambda_{\triangle\star}$. Therefore, the reason for the gradient incoherency is due to the contribution of the non-relevant documents that significantly contribute to $d_\star$ but not to $d_\triangle$. The reader may immediately recognise that setting a truncation level $\tau = +\infty$ would solve this issue at the cost of a higher computational cost. The contribution of this work pursues the following direction: to widen the set of the λ_{ij} considered still providing a limited computational cost comparable to that of a small truncation level.

Before moving forwards, let's investigate a few similar examples to further understand the behaviour of LambdaRank. Fig. 1(b) shows a similar scenario to that of Fig. 1(a), with the most relevant document $d_\triangle$ in the last position. As $d_\triangle$ moves downwards, both the RankNet cost and the discounting component of ΔNDCG ($|1/D_\triangle - 1/D_\star|$) increase generating a large upward gradient update for $d_\triangle$ and a symmetrical downward update for $d_\star$. In the setting depicted in Fig. 1(b), the gradient $\lambda_\triangle$ is larger than $\lambda_\star$ thus complying with the *Gradient Coherency* property.

Table 1: Detailed computation of LambdaRank gradients for the example illustrated in Fig. 1(a).

d_i	y_i	s_i	λ_i
$d_\star$	1	0.04	$\lambda_\star = -\lambda_{\triangle\star} + \lambda_{\star 3} + \lambda_{\star 4} + \lambda_{\star 5}$ $\approx -0.124 + 0.083 + 0.093 + 0.100 \approx 0.152$
$d_\triangle$	2	0.03	$\lambda_\triangle = +\lambda_{\triangle\star} \approx 0.124$
d_3	0	0.02	$\lambda_3 = -\lambda_{\star 3} \approx -0.083$
d_4	0	0.01	$\lambda_4 = -\lambda_{\star 4} \approx -0.093$
d_5	0	0.00	$\lambda_5 = -\lambda_{\star 5} \approx -0.100$

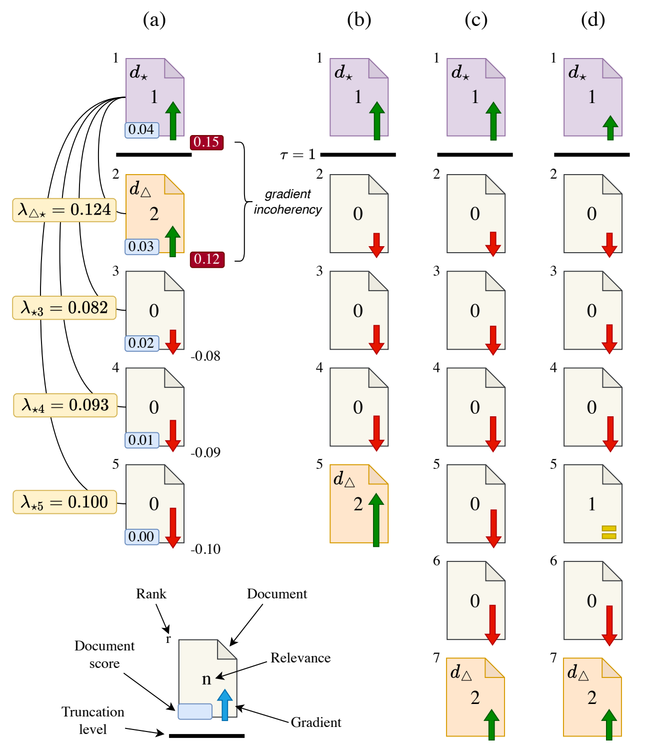


Figure 1: Examples of gradient incoherency.

However, as the rank distance between $d_\star$ and $d_\triangle$ increases, the value of $|1/\mathrm{D}_\triangle - 1/\mathrm{D}_\star|$ increases only marginally. If more non-relevant documents are placed in between $d_\star$ and $d_\triangle$ as in Fig. 1(c), the gradients generated by such non-relevant documents provide additional increments to $\lambda_\star$ but do not affect $\lambda_\triangle$. Eventually, $d_\star$ gets a larger gradient update than $d_\triangle$ breaking again the *Gradient Coherency* property. This happens because the discounting factor difference between $d_\star$ and $d_\triangle$, which should push $d_\triangle$ upwards stronger than $d_\star$, is too small to overcome the lack of gradient contribution of the pairs removed by the truncation level.

These three examples make us draw two interesting observations. First, the more the dataset contains non-relevant documents (of 0 relevance), the more the problem is likely to occur. We empirically prove this in section 5.3, especially with ISTELLA-X dataset. Second, the occurrence of gradient incoherencies is related to the model's error in a non-linear way. If the model ranks very poorly document $d_\triangle$, it is impossible for $d_\triangle$ to improve its ranking (Fig. 1(c)), if the model error is not large, the model will correctly push $d_\triangle$ upwards (Fig. 1(b)), but, if document $d_\triangle$ gets just below the truncation level, it is pushed downwards again (Fig. 1(a)). The only way document $d_\triangle$ can escape this problem is by a fortuitous update (e.g., neural network weight update (LambdaRank) or tree's leaf value (LambdaMART)), that may generate a completely different gradient since it is affected by other documents also from other queries, or by other implicit algorithm-dependent approximations.

A last scenario is illustrated in Fig. 1(d), where an additional relevant document is added by replacing document d_5 with a document with label $y_5 = 1$. Being the label the same as $d_\star$'s, the value of ΔNDCG is 0 for $\lambda_{\star5}$, so d_5 reduces the number of gradient contributions to $d_\star$. However, note that the gradient of $d_\triangle$ is not affected by d_5 because they are both beyond the truncation level. The new document makes the gradient of document $d_\star$ smaller than the gradient of document $d_\triangle$ reversing once more the gradient computation outcome.

These examples show how difficult is to model analytically the *Gradient Coherence*. We provided a few examples showing the impact of the label and the rank difference which are computed by ΔZ. Clearly, also the score difference is relevant and captured by the RankNet component of the gradient in Eq. 1. So far we focused on NDCG only. In fact, it is easy to show that the coherence property does not hold for other discount-based metrics such as Expected Reciprocal Rank [7], Rank-Biased Precision [22], etc.

Table 2: Example of computation of LambdaRank gradients with $\tau = +\infty$.

d_i	y_i	s_i	λ_i
d_1	4	0.02	$\lambda_1 = \lambda_{12} + \lambda_{13} \approx 0.176 + 0.221 \approx 0.397$
d_2	0	0.01	$\lambda_2 = -\lambda_{12} - \lambda_{32} \approx -0.176 - 0.004 \approx -0.180$
d_3	1	0.00	$\lambda_3 = -\lambda_{13} + \lambda_{32} \approx -0.221 + 0.004 \approx -0.217$

3.2 Incoherencies in LambdaRank without a truncation level

The above discussion suggests that the truncation level τ is the cause of the incoherencies in the gradient computation. Indeed, this is not true.

Let's set $\tau = +\infty$, meaning that no truncation is used, and consider the example in Tab. 2. We have three documents with scores respectively 0.02, 0.01, and 0.00, and with relevance labels respectively 4, 0, 1. Since there is no truncation level, all the pairwise gradients λ_{ij} are relevant. Document d_1 has a positive gradient λ_1 as it is ranked higher than documents with smaller relevance labels. This positive push allows gaining a desirable *margin* from the other documents. Document d_2 is non-relevant and receives a negative gradient contribution from both the other documents. Unexpectedly, document d_3, despite having an higher label than d_2, receives the strongest downward push, i.e., $\lambda_3 < \lambda_2$ with $y_3 > y_2$. This is a gradient incoherence. The reason is that swapping document d_1 with d_3 has a larger impact on the NDCG than swapping d_1 with d_2, resulting in $\lambda_{13} > \lambda_{12}$. LambdaRank prefers avoiding the risk of moving d_1 to the third position rather than pushing d_3 up to the second place. Indeed, this comes from the discount factor of the NDCG metric that demotes documents' contributions in the lower ranks. These gradients clearly push the ranking away from the ideal configuration.

This shows that LambdaRank gradients are incoherent independently of the truncation level. In this case, the major player is the discounting factor. Given the larger importance of truncated metrics, we leave the analysis of un-truncated metrics and gradients to future work. In this paper, we focus on the cases that break the *Gradient Coherence* in the presence of a truncation level, with the aim of not compromising the computational efficiency this provides.

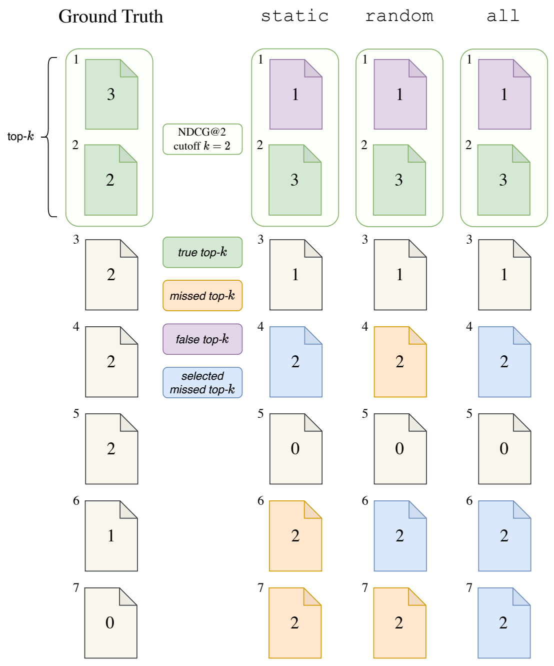


Figure 2: Lambda-*eX* *missed top-k* selection strategies.

4 LAMBDA-*EX*

We put ourselves in the scenario of truncated IR metrics optimisation. From the above, we can say that the natural choice of using a truncation level at training time to maximise a truncated metric is very beneficial in terms of computational cost, but it suffers from incoherencies in the calculation of gradients. The main contribution of this work is Lambda-*eX*, a new approach to optimise truncated ranking metrics that *limits incoherencies* while preserving *training time efficiency*. Specifically, we propose some heuristic methods to *extend* the set of document pairs considered by LambdaRank when computing gradients.

4.1 The Full-Gradient Document Set

We claim that the exacerbation of the incoherencies is due to missing computations of the λ_{ij} gradients. More specifically, relevant documents that are not ranked above the truncation level are not evaluated against all the other documents in D but only against the top-k, and this discards some of the λ_{ij} and causes under-estimation of their gradient. One possible approach is to use $I_{\tau=+\infty}$. However, this does not allow limiting the number of pairwise gradients computed to minimise the computational cost of the training process.

We thus define a *Full-Gradient Document Set* $X \subseteq D$ for which we compute a *complete* gradient estimation as in the un-truncated case $\tau = +\infty$. We compute λ_{ij} gradients as in Eq. 2 but on the basis of the set I_X:

$$I_X = \left\{(i, j) \mid d_i, d_j \in D \wedge y_i > y_j \wedge \left(d_i \in X \vee d_j \in X\right)\right\}. \quad (5)$$

The above means that the gradient λ_i of a document $d_i \in X$ is computed by considering the λ_{ij} (or λ_{ji}) for every other document $d_j \in D$. This provides a more accurate gradient estimation. We thus remark that Lambda-*eX* does not exploit a fixed truncation level, but it rather selects dynamically the set X for each different query. Indeed, the set X may not match any set I_τ for any value of τ. Also, by limiting X such that $|X| \ll |D|$, we have $|I_X| \ll |I_{\tau=+\infty}|$ and achieve a more efficient computation than in the un-truncated case.

To understand how the set X is built, let's first investigate an example from Fig. 2. The leftmost example in Fig. 2 shows a set of documents ranked according to their relevance labels. If we desire to maximise NDCG@2, the model must rank the document of relevance 3 in the first position and a document of relevance 2 in the second position. This is the ideal ranking. Suppose that a model provides the rightmost rank depicted in the same Figure. We distinguish among three kinds of documents. We call *true top-k* a *relevant* document that is ranked among the top-k and whose label occurs in the top-k of the ideal ranking. This is the case of the document in the second position of the ranking. We call *false top-k* a document that is ranked among the top-k but whose label is not among the top-k of the ideal ranking. This is the case of the top-ranked document with relevance 1. Finally, we call *missed top-k* a *relevant* document that is not ranked in the top-k but whose label is present among the top-k of the ideal ranking. This is the case of documents with label 2. Note that the above definition is not based on the document identities but rather on their relevance labels.

The previous analysis leads us to state that *missed top-k* documents receive an under-estimated gradient which may make them impossible to climb up to the top ranks. The proposed Lambda-*eX* aims at improving the learning process by providing a full and more accurate gradient estimation for the *missed top-k* documents. To do so, Lambda-*eX* may include in X the documents ranked in the top-k positions by the current model and all the *missed top-k* documents. Since the number of *missed top-k* documents can be large and we want to limit the size of X to about k, Lambda-*eX* uses some heuristic criteria to select a subset of the *missed top-k* documents to be included in X. Lambda-*eX* selection strategies are discussed in the next section.

Note that also Lambda-*eX* adopts the value ΔZ_{ij} with respect to the un-truncated metric. This means that even for a pair of documents d_i and d_j below the cutoff, the value of ΔZ_{ij} is not 0. Consequently, if $(i, j) \in I_X$ the partial derivative λ_{ij} will contribute to the gradients λ_i and λ_j.

4.2 Selection Strategies

The way Lambda-*eX* builds the set X is the core of the algorithm. We let k be the cutoff of the IR metric being optimised. First, we include in X all the top-k documents currently ranked by the model. Then, we propose three different ways to select the *missed top-k* documents ranked below the metric cutoff to be used to *extend* X. For the sake of efficiency, the first two strategies generate a set X of size $|X| \leq 2k$, while the size of X for the third strategy depends on the number of relevant documents in the query. Note that $|X|$ is query-dependent. The three selection strategies are defined as follows.

- `static`. Let h be the number of *false top-k* documents, the `static` strategy includes in X a total of h *missed top-k* documents having the largest scores. In Fig. 2 the example `static` shows how, among the possible *missed top-k* documents (in orange), it selects the best ($h = 1$) ranked documents below the cutoff among the *missed top-k* documents which have label 2. This strategy focuses on documents that are closest to the cutoff and thus most likely to fall into the problem explained in section 3.1.

- `random`. Analogous to `static`, but documents are selected randomly instead of rank-based. In example `random`, it randomly selects the second *missed top-k* of relevance 2. A random selection allows the model to see all *missed top-k* documents during training and improves model generalisation.
- `all`. Analogous to `static`, but all the *missed top-k* documents are included in X even if their number is larger than h. In example `all`, every document of relevance 2 is placed in X. In this case, all *missed top-k* documents are compared simultaneously in favour of greater generalisation of the model at the expense of the efficiency of the gradient computation.

Finally, we also propose two hybrid variants: `all-static` and `all-random`. With efficiency in mind, the goal is to limit the size of X. Depending on the query, the `all` may potentially include all the relevant documents, i.e., with a label greater than 0. To avoid such blow-up of X, the hybrid strategies roll back to either `static` or `random` in this degenerate case, otherwise, they implement the `all` strategy.

In terms of computational complexity of the gradient computation, the size of X is $k + h$ for the `static` and `random` strategies, and since $h \leq k$, it holds that $|X| \leq 2k$. Therefore, the asymptotic complexity of Lambda-*eX* with `static` or `random` in computing all the λ_i is $O(kn)$, with n the number of documents in the query. For `all`, `all-static`, and `all-random`, the *missed top-k* documents may correspond to the whole subset of relevant documents. Let n^+ be the number of such documents, the computational complexity is $O((k + n^+)n)$. Note that in general it is expected that $n^+ \ll n$, meaning a smaller cost than with $I_{\tau=+\infty}$.

5 EXPERIMENTAL EVALUATION

5.1 Datasets

We performed extensive evaluation analysis on five publicly available datasets reported in Tab. 3. Istella-X has the highest number of documents per query and non-relevant documents. MSLR Web30K Fold 1 is the most balanced since half of the documents are relevant. Instead, Yahoo! Learning to Rank Challenge Set 1 is the smallest one with about 700,000 documents and only an average of 23.73 per query. All datasets have graded relevance labels ranging from 0 to 4, where 0 stands for non-relevant and 4 for highly relevant.

5.2 Methods evaluated

It was shown empirically that embedding LambdaRank gradients within gradient-boosting decision trees, a.k.a., LambdaMART, is more effective and efficient than using LambdaRank gradients in a feed-forward artificial neural network [2]. This makes LambdaMART the state of the art in LTR. Therefore, without loss of generality, we perform all experiments and analyses by means of the more effective and efficient LambdaMART.

We refer with LambdaMART-*eX* to the strategies proposed in the previous section when applied to LambdaMART. We compare the performance of the proposed method against three baseline approaches:

- LambdaMART$_{\tau=k}$ for which the truncation level τ is set equal to the metric cutoff k. This exactly optimises the truncated metric used for the evaluation.
- LambdaMART$_{\tau=k+3}$ with $\tau = k + 3$ as suggested in [9]. A τ slightly larger than k provides a better gradient estimation for documents close to the metric cutoff at a very limited cost, without deviating too much from the evaluation metric.
- LambdaMART$_{\tau=+\infty}$ with $\tau = +\infty$ optimises the un-truncated metric, i.e., the IR metric for the whole ranking.

Table 3: Datasets properties.

Dataset	#feat.	#queries	#doc.	$\overline{\text{query len.}}$	%non-rel.
Istella-X [20]	220	10,000	26,791,447	2,679.14	99.83
Istella-S [19]	220	33,018	3,408,630	103.24	88.61
Istella-F [10]	220	33,018	10,454,629	316.63	96.29
Yahoo! Set 1 [6]	519	29,921	709,877	23.73	26.09
MSLR-30K [23]	136	31,531	3,771,125	119.60	51.47

Table 4: Hyperparameters per dataset.

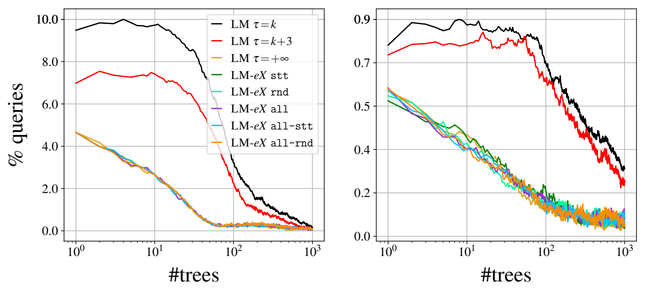
Dataset	learning_rate	num_leaves	min_data	min_hessian
Istella-X/S/F	0.05	64	20	0.001
Yahoo! Set 1	0.02	400	50	0
MSLR-30K	0.02	200	100	0

Moreover, as mentioned above, gradient incoherencies also affect other loss functions derived from LambdaLoss Framework (such as NDCG-Loss1, NDCG-Loss2, and NDCG-Loss2++ [26]). To this end, we also investigate whether extending the set of document pairs is beneficial for these loss functions. For space constraints, we focus the analysis only on NDCG-Loss2++, which is the loss function shown to achieve higher performance among the others in [26]. For the sake of clarity, hereinafter we refer to the learning algorithm that makes use of NDCG-Loss2++ loss function as LambdaLoss. We carry out all the experiments designed for LambdaMART also for LambdaLoss, so we compare LambdaLoss-*eX* with the following three baselines: LambdaLoss$_{\tau=k}$, LambdaLoss$_{\tau=k+3}$, and LambdaLoss$_{\tau=+\infty}$

We trained models through the LambdaMART implementation of the LightGBM [17] library. LambdaMART-*eX* and LambdaLoss variants were as well implemented on top of LightGBM.[1] All models were trained with the best hyperparameters found in previous works [1, 20], except for the hyperparameters of LambdaLoss which are tuned on the validation set. Tab. 4 summarises the hyperparameters used. The value of max_bin is set to 255 for all datasets and the weight coefficient μ of NDCG-Loss2++ is set to 5. For models trained with Lambda-*eX* on MSLR-30K and Yahoo! Set 1 the best value is 0.5. The hyperparameter μ manages the trade-off between the discount of NDCG-Loss2 and LambdaRank.

Models were trained to optimise NDCG for different cutoff values k: 5, 10, and 15. For each model, we stop the training process after

[1] github.com/FedericoMarcuzzi/LambdaRank-Gradients-are-Incoherent



(a) Istella-X (b) MSLR-30K

Figure 3: Percentage of queries (y-axis) affected by at least one gradient incoherence during each tree learning (x-axis). The scale in the x-axis is logarithmic.

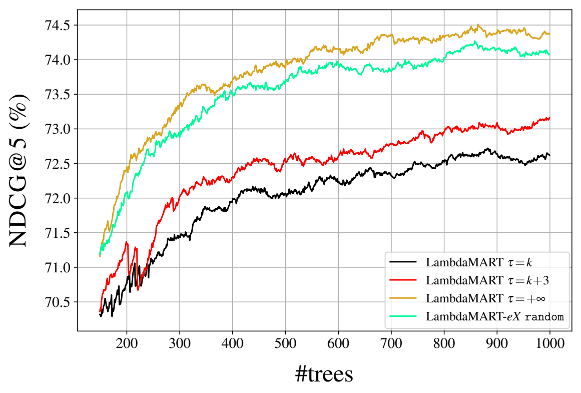


Figure 4: Models performance on Istella-X validation set.

1,000 trees and select the best iteration based on the performance achieved on the validation set.

5.3 Experimental analysis

*Does Lambda-*eX *reduce gradient incoherencies?* To answer this question we perform an analysis on the number of queries affected by gradient incoherencies during the training process. Due to space constraints, we only report the analysis based on LambdaMART (LM). Similar results are achieved with LambdaLoss. In Fig. 3 we report, for each tree of the trained forest, the number of queries encompassing at least one violation of Def.3.1 when optimising NDCG@k with $k = 5$. In fact, we restrict our attention to the most harmful violations where a *false top-k* document d_i gets a larger gradient than *missed top-k* document d_j ranked below k. We report the results of this analysis for both Istella-X and MSLR-30K. Results show that using a truncation level $\tau = k$ generates the largest amount of incoherencies, involving after 10 trees about 10% of the queries of Istella-X, 4% after 100 trees, but then falling down significantly towards the end of the forest. The initial trees of the forest are strongly affected by gradient incoherencies, which are mostly solved afterwards. Similar behaviour is exhibited by LambdaMART$_{\tau=k+3}$, with fewer incoherencies overall. This was expected since 99.83% of the documents in Istella-X are documents of relevance 0, and this increases the chance of incoherencies. The best behaviour is given by LambdaMART$_{\tau=+\infty}$ with a number of queries that quickly falls to about 2% after 10 trees. A similar trend is for MSLR-30K dataset. This confirms that discarding some of the λ_{ij} values generates a large number of incoherencies, both in LambdaMART$_{\tau=k}$ and LambdaMART$_{\tau=k+3}$.

All variants of LambdaMART-*eX* have the same behaviour of LambdaMART$_{\tau=+\infty}$. We can conclude that the proposed Lambda-*eX* succeeds in limiting the number of incoherencies, as with LambdaMART$_{\tau=+\infty}$, where all the pairwise λ_{ij} are considered.

A final interesting consideration can be made by observing Fig. 4 where the effect of having fewer incoherencies at the beginning of the training phase translates into more effective models already in the first trees of the ensemble. The first 300 trees of LambdaMART-*eX*$_{\texttt{random}}$ perform as well as 1,000 trees of LambdaMART$_{\tau=k+3}$.

*Is Lambda-*eX *more effective than truncated LambdaRank?* We evaluate the models' effectiveness in terms of NDCG@k for different cutoff values (5, 10, and 15) and we measure statistically significant improvements with respect to LambdaMART$_{\tau=k+3}$ and LambdaLoss$_{\tau=k+3}$. We choose these as reference baselines since both perform mostly better than models trained with $\tau = k$ and slightly worse than $\tau = +\infty$, but with a much lower training cost of the latter. Results are summarised in Tab. 5. In the next sub-section, we evaluate the computational cost of the discussed methods.

First of all, we highlight that the observations drawn from the results are mostly the same for both LambdaMART and LambdaLoss.

Interestingly, models trained with $\tau = +\infty$ achieve the best NDCG@k values across datasets, especially datasets from the Istella family. The only performance drops occur on MSLR-30K and Yahoo! Set 1 when LambdaLoss is used as a learning algorithm, and in these cases the performance of LambdaLoss is clearly worse than that of LambdaMART. The NDCG scores obtained with $\tau = +\infty$ might seem surprising as the target metric is not optimised, however, this highlights the effect of the gradient incoherencies intruded by $\tau = k$ and $\tau = k + 3$. Thus, considering all the document pairs provides a better gradient estimate, but this comes at a non-trivial computational cost.

Models trained with Lambda-*eX* provide very interesting results. The scored NDCG@k values are most of the times statistically significantly better than those of the baselines with $\tau = k$ and $\tau = k + 3$, and never statistically worse. Furthermore, very often they achieve the same performance as $\tau = +\infty$. Overall, Lambda-*eX* variants achieve similar performance. The `random` variant seems to achieve statistical improvements in most of the experiments, while `all` and `all-static` in a few specific cases. However, due to its lower computational complexity, the `random` strategy is preferable.

From the results in Tab. 5, two interesting considerations can be drawn about our strategy. Lambda-*eX* is most effective in datasets like the Istellas which contain many non-relevant documents. As mentioned above, the presence of many non-relevant documents increases the chance of incoherencies, which are successfully managed by Lambda-*eX*. We can conclude that Lambda-*eX* finds its best application in datasets that have many non-relevant documents. This is particularly appealing in realistic scenarios where there are far fewer documents relevant to a query than non-relevant ones.

Table 5: Methods performance in terms of NDCG (percentage). Statistically significant improvements w.r.t. $\tau = k + 3$ according to Fisher's randomisation test [13] (with a two-sided p-value) are marked with *italic* * **($p = 0.05$) and bold ** ($p = 0.01$).**

	LambdaMART			LambdaMART-*eX*					LambdaLoss			LambdaLoss-*eX*				
dataset	$\tau{=}k$	$\tau{=}k{+}3$	$\tau{=}{+}\infty$	`static`	`random`	`all`	`all-stt`	`all-rnd`	$\tau{=}k$	$\tau{=}k{+}3$	$\tau{=}{+}\infty$	`static`	`random`	`all`	`all-stt`	`all-rnd`
								NDCG@5								
Ist-X	73.32	74.11	**75.35** **	**75.19** **	**75.17** **	**75.15** **	**75.19** **	**75.17** **	74.08	74.22	**75.40** **	**75.33** **	**75.19** **	**75.14** **	**75.33** **	**75.19** **
Ist-S	70.19	70.42	*70.64* *	**70.67** **	**70.71** **	70.55	*70.65* *	*70.64* *	69.97	70.50	**71.11** **	**70.92** **	**70.92** **	**70.97** **	**70.95** **	**71.02** **
Ist-F	67.02	67.24	**67.62** **	**67.55** **	**67.67** **	**67.50** **	**67.68** **	**67.71** **	66.97	67.56	**68.18** **	**68.08** **	**68.26** **	**68.24** **	**68.07** **	**68.18** **
Yah 1	75.35	75.59	**75.85** **	75.67	75.59	75.63	75.73	75.67	75.44	75.69	74.96	75.74	75.81	75.84	75.74	75.86
MS 30	50.66	51.15	51.22	50.95	50.96	51.24	*51.42* *	51.38	50.77	51.04	49.19	50.99	50.92	51.05	51.10	51.08
								NDCG@10								
Ist-X	77.53	78.55	78.61	78.61	78.61	78.61	78.61	78.61	77.91	78.35	78.74	**78.94** **	*78.77* *	**78.94** **	**78.94** **	*78.77* *
Ist-S	76.35	76.48	**76.71** **	**76.66** **	**76.70** **	**76.69** **	**76.72** **	**76.70** **	76.63	76.89	**77.37** **	**77.26** **	**77.43** **	**77.23** **	**77.44** **	**77.28** **
Ist-F	71.85	72.07	**72.39** **	**72.42** **	**72.46** **	**72.42** **	**72.35** **	**72.46** **	72.18	72.53	**73.17** **	**73.21** **	**73.20** **	**73.16** **	**73.12** **	**73.15** **
Yah 1	79.62	79.78	79.84	79.66	79.75	79.78	79.81	79.80	79.63	79.68	79.19	**79.94** **	**79.94** **	**79.98** **	**79.93** **	*79.89* *
MS 30	52.66	53.02	52.98	52.96	53.08	*53.23* *	53.19	53.14	52.89	53.08	51.36	52.99	52.98	52.95	53.02	52.99
								NDCG@15								
Ist-X	79.00	79.29	79.45	79.44	79.48	79.44	79.44	79.48	78.81	79.06	**79.73** **	*79.60* *	**79.78** **	*79.60* *	*79.60* *	**79.78** **
Ist-S	80.63	80.59	*80.73* *	80.69	*80.71* *	**80.75** **	**80.80** **	*80.73* *	80.96	81.15	81.29	*81.31* *	**81.40** **	**81.38** **	**81.38** **	**81.38** **
Ist-F	75.46	75.56	**75.87** **	**75.94** **	**75.90** **	**75.92** **	**76.00** **	**76.00** **	75.97	76.24	**76.74** **	**76.85** **	**76.85** **	**76.83** **	**76.82** **	**76.85** **
Yah 1	82.01	81.96	82.03	81.94	82.07	82.04	82.07	82.04	81.88	81.95	81.50	*82.15* *	**82.16** **	82.09	82.09	82.09
MS 30	54.60	54.72	54.67	54.82	**54.93** **	54.84	54.75	54.83	54.68	54.60	53.23	54.63	54.65	54.64	54.69	54.62

Table 6: Training time expressed in milliseconds.

		LambdaMART			LambdaMART-*eX*				
k	dataset	$\tau{=}k$	$\tau{=}k{+}3$	$\tau{=}{+}\infty$	`stt`	`rnd`	`all`	`a-stt`	`a-rnd`
					training time per objective function				
	Ist-X	89	100	2574	115	132	118	113	130
	Ist-S	12	16	31	17	20	18	18	22
5	Ist-F	30	38	119	43	49	46	45	52
	Yah 1	3	4	12	4	6	6	6	7
	MS 30	4	5	34	5	7	9	8	10
					training time per tree				
	Ist-X	672	751	3253	792	815	795	796	813
	Ist-S	156	122	143	129	129	130	130	133
5	Ist-F	263	288	380	306	308	301	299	307
	Yah 1	207	205	215	208	294	287	209	212
	MS 30	291	298	388	324	317	326	314	334

The second interesting consideration we can draw is that as the metric cutoff increases, the performance gap between Lambda-*eX* and the baselines decreases. The reason behind it is straightforward. Recall that in the experiments we fixed τ equal to the cutoff k and to k+3. Thus, with a small cutoff the probability of a relevant document being ranked below the truncation level is large, consequently many λ_{ij} are discarded. As the cutoff increases, it is more likely that relevant documents are ranked above a larger truncation level and therefore their gradient is fully computed.

*Is Lambda-*eX *more efficient than LambdaRank?* The last analysis we perform concerns efficiency in terms of training time. In particular, we measure the average training time spent by the LightGBM library in training a tree and in executing the objective function (computing the gradient for each document of each query) during the training of a single tree. We run this analysis when optimising NDCG@5. Results are reported in Tab. 6.

The average execution time of an iteration of LambdaMART-*eX*'s objective function aligns with the one of LambdaMART$_{\tau=k}$ and LambdaMART$_{\tau=k+3}$. The reason behind this relies on a similar asymptotic complexity. As expected, LambdaMART$_{\tau=+\infty}$ is the one that spends more time executing the objective function since it has to process $O(n^2)$ document pairs. Note that the cost of LambdaMART$_{\tau=+\infty}$ can be up to 30 times larger.

The difference in execution time of the objective functions affects the training of a tree only with long results lists. This can be seen with Istella-X which tree learning time increases from 792 milliseconds with LambdaMART-*eX*$_{\texttt{static}}$ to 3,253 milliseconds with LambdaMART$_{\tau=+\infty}$. This is a 4× slowdown that happens for each of the 1,000 trees of the model trained.

Another interesting observation in favour of LambdaMART-*eX*'s efficiency is that it manages to achieve the same performance as LambdaMART$_{\tau=k}$ and LambdaMART$_{\tau=k+3}$ with far fewer trees. In Fig. 4, the model trained with LambdaMART-*eX*$_{\texttt{random}}$ achieves the same performance as LambdaMART$_{\tau=k}$ with about 250 trees in Istella-X, and the same performance as LambdaMART$_{\tau=k+3}$ with 300 trees. The difference in model size significantly reduces training time, even though the average time taken to train a single tree is similar. The same behaviour was observed for all the Lambda-*eX* variants. Note that since the computational complexity of LambdaMART is exactly the same as LambdaLoss, the above results and considerations generalise for LambdaLoss as well.

In conclusion, LambdaMART-*eX* can reduce the training time compared to LambdaMART$_{\tau=k}$ and LambdaMART$_{\tau=k+3}$ by training equally effective models with far fewer trees, and compared to LambdaMART$_{\tau=+\infty}$ especially when training datasets with a high average number of documents per query.

6 RELATED WORK

One of the biggest challenges in Learning to Rank is metrics optimisation. Ranking metrics are non-differentiable since they are inherently tied to the order of the documents. Therefore, unlike most machine learning techniques, an objective function that makes use of IR metrics cannot be directly optimised by gradient descent methods. However, effective ranking models are essential in a huge variety of applications in IR systems. This began an arms race to address the problem from multiple directions. Among those attempts are ranking metrics approximation or loss function that indirectly aligns with the desired metric. However, most LTR approaches optimise a loss function that is loosely related to a ranking metric or is its upper bound.

A well-known class of solutions that aims to minimise the number of errors committed in ranking pairs of documents is that of pairwise approaches. These include RankNet [4], a pairwise approach that optimises a probabilistic loss function by mapping the model output to a learned probability. Another well-known approach is AdaRank [28], which learns weak rankers that minimise the pairwise misranking error, and then linearly combines them together for prediction. Pairwise approaches typically optimise a convex upper-bounds of the pair misranking error. However, this sort of optimisation does not directly imply an improvement in the ranking metric thus leading to a mismatch between model optimisation and effectiveness on the desired metric.

To fill the gap, listwise approaches embed the information on the status of the entire ranking list into the optimisation process. Listwise approaches fall mainly into two macro-categories, those that approximate the ranking metric through a smooth surrogate such as SoftRank [25] and ApproxNDCG [24], and those that use heuristics to construct a smooth surrogate loss function such as ListNET [5], $\mathrm{XE}_{\mathrm{NDCG}}$ [1], and LambdaRank [2]. ListNET minimises the cross-entropy between ground truth and model score distribution. $\mathrm{XE}_{\mathrm{NDCG}}$ is a cross-entropy loss function that guarantees strong theoretical properties like optimising a convex bound on mean NDCG.

Finally, LambdaRank is an approach initially designed for artificial neural networks, which does not try to optimise a loss function but heuristically defines the loss gradient. LambdaRank is one of the most effective approaches among the listed above and its variant called LambdaMART [2, 27] which makes use of gradient boosting trees is considered the state of the art in LTR. About a decade later in [26] authors proposed a probabilistic framework for ranking metric optimisation called LambdaLoss [26] and showed how LambdaRank is a special configuration with a well-defined loss. In the article, they defined different metric-driven loss functions, based on NDCG and ARP [16]. Among them, NDCG-Loss2 and NDCG-Loss2++ obtained the most statistically significant results. In particular, NDCG-Loss2 is a metric-driven loss function that shares many similarities with LambdaRank. The difference lies in the definition of the discount used in ΔNDCG. Specifically, the discount $\rho_{ij} = |1/\mathrm{D}_i - 1/\mathrm{D}_j|$ in Eq. 3 becomes $\delta ij = |1/\mathrm{D}_{|i-j|} - 1/\mathrm{D}_{|i-j|+1}|$. Finally, the loss function NDCG-Loss2++ is a linear combination of the two discounting $\rho_{ij} + \mu\delta_{ij}$, with μ a weight coefficient managing the trade-off between the two.

Another interesting observation derives from [12], where the authors investigated whether training a model on the same truncated metric used for the evaluation (e.g., NDCG@k) is better than training it on the un-truncated metric (e.g., NDCG). What emerged is in line with our results, training on un-truncated metrics returns better-performing models. However, the reason given by [12] only tells half the story. They claim that optimising the truncated metric reduces the number of contributions λ_{ij} each gradient λ_i receives. They also showed that with an equal number of document pairs (of λs) during training same performance as the un-truncated metric can be obtained. However, in this work, we show how with the same order of magnitude in the number of pairs as in the truncated metric we can obtain the same performance as the un-truncated metric. This confirms that the reason behind the drop in performance lies in the exacerbation of the gradient incoherencies and that by selecting the right pairs of documents it is possible to obtain the same performance as the un-truncated metric.

Document sampling strategies like SelGB [20], SOUR [21], and SGB [14] are also worth mentioning. These try to improve the effectiveness/efficiency of the model by sampling/removing good/bad relevant/non-relevant examples from the training set. It is important to highlight why our solution is not a document sampling strategy. These sampling approaches discard some documents from (some iterations of) the training. The proposed Lambda-*eX* does not remove any document, but rather it selects a few documents for which a complete gradient computation is needed. Document sampling approach selects documents randomly, or with a preference for hard negatives, but they do not encompass any notion of gradient coherency.

Finally, deep learning approaches, such as those based on BERT [11], RoBERTa [18], and ELECTRA [8], are not within the scope of this work. On the one hand, they rely on the availability of full text, on the other hand, they are usually limited to point-wise or list-wise losses. In general, the approach proposed in this work generalises to any machine learning model exploiting LambdaRank gradients.

7 CONCLUSION

We have shown that LambdaRank and derivatives (such as LambdaLoss) are affected by gradient incoherencies, exacerbated when optimising truncated metrics. We designed a new approach called Lambda-*eX* to counter this phenomenon while keeping the training focused on the metric to be optimised, without affecting the efficiency of the algorithm. Through extensive experiments, we have shown that LambdaMART-*eX* can achieve statistically significant improvement in terms of NDCG@k w.r.t. models trained to directly optimise the target metric (such as $\mathrm{LambdaMART}_{\tau=k}$ and $\mathrm{LambdaMART}_{\tau=k+3}$) while maintaining its efficiency in terms of training time. Furthermore, we demonstrated that the same improvement can also be achieved by LambdaMART's derivatives, such as LambdaLoss. Finally, when Lambda-*eX* is used to train models on datasets with a high average number of documents per query, it is able to achieve the same performance as the original definition of LambdaMART (and LambdaLoss) while significantly reducing the training time, e.g., about 40 minutes difference in Istella-X.

ACKNOWLEDGEMENT

This work was partially supported by both SERICS project under the NRRP M4C2 Inv.1.3 PE00000014 and iNEST project under the NRRP M4C2 Inv.1.5 ECS00000043, funded by the EU - NGEU. This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

[1] Sebastian Bruch. 2021. An Alternative Cross Entropy Loss for Learning-to-Rank. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 118–126. https://doi.org/10.1145/3442381.3449794

[2] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[3] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. 2006. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, Bernhard Schölkopf, John C. Platt, and Thomas Hofmann (Eds.). MIT Press, 193–200. https://proceedings.neurips.cc/paper/2006/hash/af44c4c56f385c43f2529f9b1b018f6a-Abstract.html

[4] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005 (ACM International Conference Proceeding Series, Vol. 119)*, Luc De Raedt and Stefan Wrobel (Eds.). ACM, 89–96. https://doi.org/10.1145/1102351.1102363

[5] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007 (ACM International Conference Proceeding Series, Vol. 227)*, Zoubin Ghahramani (Ed.). ACM, 129–136. https://doi.org/10.1145/1273496.1273513

[6] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to Rank Challenge Overview. In *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010, Haifa, Israel, June 25, 2010 (JMLR Proceedings, Vol. 14)*, Olivier Chapelle, Yi Chang, and Tie-Yan Liu (Eds.). JMLR.org, 1–24. http://proceedings.mlr.press/v14/chapelle11a.html

[7] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, David Wai-Lok Cheung, Il-Yeol Song, Wesley W. Chu, Xiaohua Hu, and Jimmy Lin (Eds.). ACM, 621–630. https://doi.org/10.1145/1645953.1646033

[8] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=r1xMH1BtvB

[9] Microsoft Corporation. 2023. *LightGBM Release 3.3.3.99.* https://readthedocs.org/projects/lightgbm/downloads/pdf/latest/

[10] 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 (2016), 15:1–15:31. https://doi.org/10.1145/2987380

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[12] Pinar Donmez, Krysta M. Svore, and Christopher J. C. Burges. 2009. On the local optimality of LambdaRank. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, James Allan, Javed A. Aslam, Mark Sanderson, ChengXiang Zhai, and Justin Zobel (Eds.). ACM, 460–467. https://doi.org/10.1145/1571941.1572021

[13] R.A. Fisher. 1935. *The design of experiments. 1935.* Oliver and Boyd, Edinburgh.

[14] Jerome H. Friedman. 2002. Stochastic Gradient Boosting. *Comput. Stat. Data Anal.* 38, 4 (feb 2002), 367–378. https://doi.org/10.1016/S0167-9473(01)00065-2

[15] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20, 4 (2002), 422–446. https://doi.org/10.1145/582415.582418

[16] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*, Maarten de Rijke, Milad Shokouhi, Andrew Tomkins, and Min Zhang (Eds.). ACM, 781–789. https://doi.org/10.1145/3018661.3018699

[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: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 3146–3154. https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[19] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. 2016. Post-Learning Optimization of Tree Ensembles for Efficient Ranking. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel (Eds.). ACM, 949–952. https://doi.org/10.1145/2911451.2914763

[20] Claudio Lucchese, Franco Maria Nardini, Raffaele Perego, Salvatore Orlando, and Salvatore Trani. 2018. Selective Gradient Boosting for Effective Learning to Rank. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz (Eds.). ACM, 155–164. https://doi.org/10.1145/3209978.3210048

[21] Federico Marcuzzi, Claudio Lucchese, and Salvatore Orlando. 2022. Filtering out Outliers in Learning to Rank. In *ICTIR '22: The 2022 ACM SIGIR International Conference on the Theory of Information Retrieval, Madrid, Spain, July 11 - 12, 2022*, Fabio Crestani, Gabriella Pasi, and Éric Gaussier (Eds.). ACM, 214–222. https://doi.org/10.1145/3539813.3545127

[22] Alistair Moffat and Justin Zobel. 2008. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.* 27, 1 (2008), 2:1–2:27. https://doi.org/10.1145/1416950.1416952

[23] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[24] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Inf. Retr.* 13, 4 (2010), 375–397. https://doi.org/10.1007/s10791-009-9124-x

[25] Michael J. Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. SoftRank: optimizing non-smooth rank metrics. In *Proceedings of the International Conference on Web Search and Web Data Mining, WSDM 2008, Palo Alto, California, USA, February 11-12, 2008*, Marc Najork, Andrei Z. Broder, and Soumen Chakrabarti (Eds.). ACM, 77–86. https://doi.org/10.1145/1341531.1341544

[26] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, 1313–1322. https://doi.org/10.1145/3269206.3271784

[27] Qiang Wu, Christopher J. C. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Inf. Retr.* 13, 3 (2010), 254–270. https://doi.org/10.1007/s10791-009-9112-1

[28] Jun Xu and Hang Li. 2007. AdaRank: a boosting algorithm for information retrieval. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*, Wessel Kraaij, Arjen P. de Vries, Charles L. A. Clarke, Norbert Fuhr, and Noriko Kando (Eds.). ACM, 391–398. https://doi.org/10.1145/1277741.1277809