Computationally Efficient Optimization of Plackett-Luce Ranking Models for Relevance and Fairness

Harrie Oosterhuis
Radboud University
Nijmegen, The Netherlands
harrie.oosterhuis@ru.nl

ABSTRACT

Recent work has proposed stochastic Plackett-Luce (PL) ranking models as a robust choice for optimizing relevance and fairness metrics. Unlike their deterministic counterparts that require heuristic optimization algorithms, PL models are fully differentiable. Theoretically, they can be used to optimize ranking metrics via stochastic gradient descent. However, in practice, the computation of the gradient is infeasible because it requires one to iterate over all possible permutations of items. Consequently, actual applications rely on approximating the gradient via sampling techniques.

In this paper, we introduce a novel algorithm: PL-Rank, that estimates the gradient of a PL ranking model w.r.t. both relevance and fairness metrics. Unlike existing approaches that are based on policy gradients, PL-Rank makes use of the specific structure of PL models and ranking metrics. Our experimental analysis shows that PL-Rank has a greater sample-efficiency and is computationally less costly than existing policy gradients, resulting in faster convergence at higher performance. PL-Rank further enables the industry to apply PL models for more relevant and fairer real-world ranking systems.

CCS CONCEPTS

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

KEYWORDS

Learning to Rank; Policy Gradients; Ranking Metric Optimization

ACM Reference Format:
Harrie Oosterhuis. 2021. Computationally Efficient Optimization of Plackett-Luce Ranking Models for Relevance and Fairness. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462830

1 INTRODUCTION

Learning to Rank (LTR) is a branch of machine learning that covers methods for optimizing ranking systems [18]. As a result, LTR is very important for search and recommendation applications that heavily depend on well-functioning ranking systems. These systems go through large collections of items and produce a ranking: a small ordered set of items. A good ranking can make it very easy for the user to find the items they are looking for, even when these items are part of a very large collection [8, 10, 26].

Traditionally, ranking systems consist of a scoring function that assigns an individual score to each item, and subsequently, produce rankings by sorting items according to their assigned scores [6, 17, 18, 31]. The crucial difference with LTR and regression or classification is that only the relative differences between scores matter. In other words, in LTR it is not important what the exact score of an item is but how much greater or smaller it is than the scores of the other items. The main difficulty in LTR is that the ranking procedure is deterministic and non-differentiable, since there is no gradient w.r.t. the sorting function. The methods in the LTR field can be divided in applying one of two solutions: optimizing a heuristic function that bounds or approximates the ranking performance [4–6, 17, 18, 31]; or optimizing a probabilistic ranking system [7, 21, 29, 34].

In recent years, the popularity of the Plackett-Luce (PL) ranking model has increased [6, 11, 19, 25, 28]. It models ranking as a succession of decision problems where each individual decision is made by a PL model (also known as the *Soft-Max* in deep learning). Previous research from the industry indicates that the probabilistic nature of the PL model leads to more robust performance [3]. In online LTR, it appears the PL model is very good at exploration because it explicitly quantifies its uncertainty [21, 23]. Recent work has also posed that the PL model is well suited to address fairness aspects of ranking [11, 28], because unlike deterministic models, it can give multiple items an equal probability of being the top-item.

However, calculating the gradient of a PL ranking model requires an iteration over every possible ranking that the model could produce, i.e., every possible permutation. In practice this computational infeasibility is circumvented by estimating the gradient based on rankings sampled from the model [11, 22, 23, 28]. The main downside of this approach is that it can be computationally very costly. This is a particular problem in online settings where optimization is performed periodically as more data is gathered [20, 22, 23, 28].

In this paper, we introduce PL-Rank a novel method that can efficiently optimize both relevance and exposure-based fairness ranking metrics or linear combinations of them. We contribute to the theory of the LTR field, by deriving novel estimators that can unbiasedly estimate the gradient of a PL ranking model w.r.t. a ranking metric, on which PL-Rank is build. To the best of our knowledge, PL-Rank is the first LTR method that utilizes specific properties of ranking metrics and the PL-ranking model. Our experimental results show that compared to existing LTR methods, PL-Rank has increased sample-efficiency: it requires less sampled rankings to reach the same performance, and increased computational time-efficiency: PL-Rank requires less time to compute the estimation of the gradient and less computational time to converge at optimal performance.

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07.
https://doi.org/10.1145/3404835.3462830

The introduction of PL-Rank makes the optimization of PL ranking models more practical by greatly reducing its computational costs, additionally, these gains also help in the further promotion of fairness aspects of ranking models [11].

2 RELATED WORK

One of the earliest LTR approaches is the pairwise approach where the loss function is based on the order of pairs of items [5, 17, 18]. While pairwise losses are easy to compute and scale well with the number of items to rank, pairwise loss functions do not consider the entire ranking [6, 18]. As a result, minimizing a pairwise loss often does not translate to the optimal ranking behavior. Subsequently, the idea of a listwise method that considers the complete ranking was introduced with the ListNet and ListMLE methods [7, 34]. These methods optimize PL ranking models to maximize the probability of the optimal ranking. They have three main limitations: (i) They assume there is a single optimal ranking per query, while often there are multiple optimal rankings. (ii) They are not based on actual ranking metrics and thus may not actually maximize the desired metrics over the entire dataset. (iii) They bring substantial computational costs, i.e. the cost of ListNet is so high Cao et al. only optimize the top-1 ranking [7]. Some of these issues are avoided by the later *LambdaRank* and *LambdaMART* methods [6]. LambdaRank is an extension of the pairwise *RankNet* method, where the loss function weights each pair by the absolute difference in Discounted Cumulative Gain (DCG) that would result from swapping the pair. This approach optimizes a deterministic ranking model and also works for other ranking metrics than DCG [6, 31]. The Lambda methods are listwise because their gradient is based on the ranking metric values of the current ranking and therefore consider the complete ranking. While initially, there was only empirical evidence for the great performance of Lambda methods in optimizing ranking metrics [6, 12]. Recently, Wang et al. [31] proved that the Lamba methods optimize a lower bound on ranking metrics and introduced a novel bound in the form of the LambdaLoss framework [31]. Thus although the Lamba methods are based around ranking metrics and are computationally feasible, they do not optimize metrics directly and are therefore heuristic methods. Another heuristic approach is to replace the rank function in a metric by a differentiable probabilistic approximation, notable examples of this approach are *SoftRank* [29] and *ApproxNDCG* [4]. While all of these methods can be useful in practice, none optimize rankings metrics directly in a computationally feasible manner.

Interestingly, multiple lines of previous work have found PL-ranking models to be very effective for various ranking tasks: for result randomization in interleaving [16], multileaving [27] and counterfactual evaluation [22]; for exploration in online LTR [21, 23]; for fair distributions of attention exposure [11, 28]; and for topic diversity in ranking [32, 35]. In particular, Bruch et al. [3] argue that the stochastic nature of PL models results in more robust ranking performance. Furthermore, Bruch et al. show that, with small alterations, many existing LTR methods can adequately optimize PL methods. An interesting property of the PL ranking model is that it has a gradient w.r.t. ranking metrics but that it is generally infeasible to compute, existing work has thus approximated this gradient [11, 22, 23, 28] (see Section 4). The computational costs are particularly relevant because the PL model is often used in online settings where it optimization is performed repeatedly and frequently [22, 23, 28]. For instance, Oosterhuis and de Rijke [23] show that frequently optimizing the logging-policy model during the gathering of data greatly reduces the data-requirements for online/counterfactual LTR. Similarly, Morik et al. [20] show that to prevent very unfair distributions of exposure, rankings should be updated continuously as more interaction data is gathered. To the best of our knowledge, no previous work has developed a novel LTR method specifically for PL ranking models that optimizes ranking metrics directly nor with a focus on computational efficiency.

3 RELEVANCE RANKING METRICS

Generally, LTR methods assume each item d has some *relevance* w.r.t. a query q [18], in the context of fairness this is often considered the *merit* of an item [11]. This is often modelled as the probability that a user finds the item d relevant for their issued query q. To keep our notation brief, we use ρ_d to denote this probability: $P(R=1|q,d)=\rho_d$. Whenever we talk about the relevance of an item, it will be clear from the context what the corresponding query is, hence we keep q out of our notation for the sake of brevity.

Rankings are ordered lists of items, we use y to denote a ranking and y_k for the kth item in ranking y: $y=[y_1,y_2,...,y_K]$, thus $y_k=d$ means that d is the item at rank k in ranking y. We will assume that all rankings are of length K, for instance, because only K items can be displayed. A ranking model π can be seen as a distribution over rankings, where $\pi(y \mid q)$ indicates the probability that ranking y is sampled for query q by model π. For brevity, we will use $\pi(y)=\pi(y \mid q)$ as the corresponding query will always be clear from the context.

The relevance performance (also called the reward) of a ranking model is represented by the metric value $\mathcal{R}$. Relevance ranking metrics use weights per rank θ where the relevance of an item at rank k is weighted by θ_k. For a single query, the metric is computed as an expectation over the ranking behavior of π:

$$\begin{aligned}\mathcal{R}(q) &= \sum_{y\in\pi}\pi(y|q)\sum_{k=1}^{K}\theta_k P(R=1|q,y_k)\\ &= \sum_{y\in\pi}\pi(y)\sum_{k=1}^{K}\theta_k\rho_{y_k} = \mathbb{E}_y\left[\sum_{k=1}^{K}\theta_k\rho_{y_k}\right].\end{aligned} \tag{1}$$

By choosing θ_k accordingly, $\mathcal{R}(q)$ can represent the most common relevance ranking metrics. For example, top-K DCG is computed with the following weights: $\theta_k^{\text{DCG@K}}=\frac{\mathbb{1}[k\leq K]}{\log_2(k+1)}$, precision at K with: $\theta_k^{\text{PREC@K}}=\frac{1}{K}\mathbb{1}[k\leq K]$, or Average Relevance Position (ARP) can be represented by: $\theta_k^{\text{ARP}}=-k$. The overall relevance of a ranking system π is simply the expected performance over the distribution of user-issued queries:

$$\mathcal{R}=\mathbb{E}_q[\mathcal{R}(q)]=\sum_{q\in Q}P(q)\mathcal{R}(q). \tag{2}$$

The value of $\mathcal{R}$ is also often called the *performance* or the *reward*. Accordingly, LTR for relevance optimizes π to maximize $\mathcal{R}$, given the item relevances ρ_d and according to the chosen metric represented by θ_k.

4 PLACKETT-LUCE RANKING MODELS

As noted in Section 2, the PL model [19, 25] has often been deployed to model a probabilistic distribution over rankings [3, 11, 16, 21–23, 28, 32, 35]. In the PL model, an item is chosen from a pool of available items based on the individual scores each item has. For our ranking problem, a learned prediction model m predicts the log score of an item d w.r.t. to query q as $m(q,d) \in \mathbb{R}$. For brevity, we again keep the query out of our notation: $m(q,d) = m(d)$.

The probability that item d is chosen to be the kth item in ranking y from the set of items $\mathcal{D}$ is the score of d: $e^{m(d)}$, divided by the sum of scores for the items that have not been placed yet:

$$\pi(d|y_{1:k-1},\mathcal{D}) = \frac{e^{m(d)}\mathbb{1}\left[d \notin y_{1:k-1}\right]}{\sum_{d' \in D \setminus y_{1:k-1}} e^{m(d')}}, \tag{3}$$

where $y_{1:k-1}$ indicates the ranking up to rank $k-1$, i.e.,

$$y_{1:k-1} = [y_1, y_2, ..., y_{k-1}]. \tag{4}$$

As such the placement probabilities at k depend only on the scores of the items not placed before rank k. Because the learned m predicts the log score, the actual score is always greater than zero: $e^{m(d)} > 0$, consequently, $\pi(d|y_{1:k-1})$ is always a valid probability distribution over the unplaced items. We note that this probability is extremely similar to the Soft-Max function commonly used in deep learning. To prevent items from appearing in a ranking twice, the probability of $\pi(d|y_{1:k-1},\mathcal{D}) = 0$ if d has already been placed: $d \in y_{1:k-1}$.

Finally, the probability of a ranking is simply the product of the placement probabilities of each individual item:

$$\pi(y) = \prod_{k=1}^{K} \pi(y_k|y_{1:k-1},\mathcal{D}). \tag{5}$$

4.1 Computationally Efficient Sampling

An advantage of the PL ranking model is that rankings can be sampled quite efficiently. At first glance, sampling a ranking may seem computationally costly, as it involves repeatedly sampling from the item distribution and renormalizing it. However, using the Gumbel Softmax trick [14] one can sample an entire ranking without having to calculate any of the actual probabilities [3].

Our goal is to acquire a sampled ranking $y^{(i)}$ from the π distribution: $y^{(i)} \sim \pi$. Instead of calculating the actual placement probabilities, for each item a sample from the Gumbel distribution is taken: $\gamma_d^{(i)} \sim \text{Gumbel}(0,0)$. This can be done by first sampling uniformly from the [0,1] range: $\zeta_d^{(i)} \sim \text{Uniform}(0,1)$, and then applying: $\gamma_d^{(i)} = -\log(-\log(\zeta_d^{(i)}))$. Subsequently, per item we take the sum of their Gumbel sample and their log score:

$$\hat{m}_d^{(i)} = m(d) + \gamma_d^{(i)}. \tag{6}$$

Finally, we sort the items according to their $\hat{m}^{(i)}$ values, resulting in the sampled ranking:

$$\begin{aligned} y^{(i)} &= [y_1^{(i)}, y_2^{(i)}, ..., y_K^{(i)}] \\ &\text{s.t. } \forall (y_x^{(i)}, y_z^{(i)}), \quad x < z \rightarrow \hat{m}_{y_x}^{(i)} \geq \hat{m}_{y_z}^{(i)}. \end{aligned} \tag{7}$$

This sampling procedure follows the PL distribution of π [3, 14]. In practice, this means we can sample rankings as quickly as we can sort top-K rankings, which translates to a computational complexity of $O(|\mathcal{D}|\log(|\mathcal{D}|))$.

4.2 Basic Policy Gradient Estimation

As noted by Singh and Joachims [28] and Bruch et al. [3], PL ranking models can be optimized via policy-gradients. They utilize the famous *log-trick* from the *REINFORCE* algorithm [33]. We apply the log-trick to Eq. 5 to obtain:

$$\frac{\delta}{\delta m}\pi(y) = \pi(y)\left[\frac{\delta}{\delta m}\log(\pi(y))\right]. \tag{8}$$

By combining this result with Eq. 1, we find that the derivate can be expressed as an expectation over the ranking distribution π:

$$\begin{aligned} \frac{\delta}{\delta m}\mathcal{R}(q) &= \sum_{y \in \pi}\left[\frac{\delta}{\delta m}\pi(y)\right]\sum_{k=1}^{K}\theta_k \rho_{y_k} \\ &= \sum_{y \in \pi}\pi(y)\left[\frac{\delta}{\delta m}\log(\pi(y))\right]\left(\sum_{k=1}^{K}\theta_k \rho_{y_k}\right) \\ &= \mathbb{E}_y\Bigg[\underbrace{\left[\frac{\delta}{\delta m}\log(\pi(y))\right]}_{\text{gradient w.r.t. complete ranking}}\underbrace{\left(\sum_{k=1}^{K}\theta_k \rho_{y_k}\right)}_{\text{full reward}}\Bigg]. \end{aligned} \tag{9}$$

We see that this policy gradient is composed of two parts: a gradient w.r.t. the log probability of a complete ranking multiplied by the reward for that ranking. In practice, it is infeasible to compute this gradient exactly since it requires a summation over every possible ranking y. Luckily, because the gradient can be expressed as an expectation over ranking y w.r.t. to the distribution according to π, the gradient can be estimated using a simple sampling strategy. If we sample N rankings from π for query q, with $y^{(i)}$ denoting the ith sample, then the gradient can be estimated using:

$$\frac{\delta}{\delta m}\mathcal{R}(q) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\delta}{\delta m}\log(\pi(y^{(i)}))\right]\left(\sum_{k=1}^{K}\theta_k \rho_{y_k^{(i)}}\right). \tag{10}$$

A straightforward implementation first samples N rankings using Gumbel sampling (Section 4.1), and then computes the reward for each ranking, to finally use a machine learning framework to compute the gradient w.r.t. the log probabilities: $\left[\frac{\delta}{\delta m}\log(\pi(y^{(i)}))\right]$. This approach works well with currently popular deep-learning frameworks such as *PyTorch* [24] or *Tensorflow* [1].

This concludes our description of the basic policy gradient approach to optimizing PL ranking models. While this approach works adequately, our results show that this approach is computationally expensive and can have convergency issues when $N < 1000$. Finally, we note that this approach is not specific to PL-ranking models as it essentially just applies the very general *REINFORCE* algorithm [33]. In contrast, the remainder of this paper will introduce methods that make use of specific PL properties, and as a result, show better performance in our experimental results.

5 METHOD: PL-RANK FOR RELEVANCE

In this section, we will derive three novel methods for estimating the gradient of a PL-ranking model. The latter two estimators make our proposed PL-Rank method, the former is an intermediate step

between the basic policy gradient estimation and PL-Rank. Unlike existing methods, PL-Rank utilizes specific properties about PL ranking models and ranking metrics.

5.1 Ranking Metric Based Approximation

The basic estimator in Eq. 10 only deals with the reward of the entire ranking. This can lead to very unintuitive behavior, for instance, when a ranking is sampled that receives a very high reward but only due the first placed item, the gradient w.r.t. entire ranking will be multiplied with this reward. Thus despite the fact that only the first item contributed positively to the reward, the probability of placement for all items will be increased.

By rewriting Eq. 1 we can see that relevance rewards only need to interact with the probability of the ranking up to the corresponding rank:

$$\begin{aligned}\mathcal{R}(q) &= \sum_{y\in\pi}\pi(y)\sum_{k=1}^{K}\theta_k\rho_{y_k} = \sum_{k=1}^{K}\theta_k\sum_{y\in\pi}\pi(y)\rho_{y_k} \\ &= \sum_{k=1}^{K}\theta_k\sum_{y_{1:k}\in\pi}\pi(y_{1:k})\rho_{y_k},\end{aligned} \tag{11}$$

where $\sum_{y_{1:k}\in\pi}$ is a summation over all possible (sub)rankings of length k according to π. In other words, the relevance ρ_{y_k} at any rank k only interacts with the probability of the ranking up to k: $\pi(y_{1:k})$. Inuitively this makes sense because the placement of any item after k will not affect the previously obtained reward. We can use this fact when estimating the gradient w.r.t. the complete reward.

Before we derive the gradient w.r.t. the complete reward, we first consider that the derivate of the log probability of a ranking can be decomposed as a sum over log probabilities of the individual item placements. Using Eq. 5:

$$\begin{aligned}\left[\frac{\delta}{\delta m}\pi(y_{1:k})\right] &= \pi(y_{1:k})\left[\frac{\delta}{\delta m}\log(\pi(y_{1:k}))\right] \\ &= \pi(y_{1:k})\sum_{x=1}^{k}\left[\frac{\delta}{\delta m}\log(\pi(y_x\,|\,y_{1:x-1}))\right].\end{aligned} \tag{12}$$

We can now use to get the derivative w.r.t. to $\mathcal{R}(q)$ using Eq. 11 & 12:

$$\begin{aligned}\frac{\delta}{\delta m}\mathcal{R}(q) &= \sum_{k=1}^{K}\theta_k\sum_{y_{1:k}\in\pi}\rho_{y_k}\left[\frac{\delta}{\delta m}\pi(y_{1:k})\right] \\ &= \sum_{k=1}^{K}\theta_k\sum_{y_{1:k}\in\pi}\pi(y_{1:k})\rho_{y_k}\sum_{x=1}^{k}\left[\frac{\delta}{\delta m}\log(\pi(y_x|y_{1:x-1}))\right] \\ &= \sum_{k=1}^{K}\mathbb{E}_{y_{1:k}}\left[\theta_k\rho_{y_k}\sum_{x=1}^{k}\left[\frac{\delta}{\delta m}\log(\pi(y_x\,|\,y_{1:x-1}))\right]\right] \\ &= \mathbb{E}_y\left[\sum_{k=1}^{K}\theta_k\rho_{y_k}\sum_{x=1}^{k}\left[\frac{\delta}{\delta m}\log(\pi(y_x\,|\,y_{1:x-1}))\right]\right] \\ &= \mathbb{E}_y\Bigg[\underbrace{\sum_{k=1}^{K}\left[\frac{\delta}{\delta m}\log(\pi(y_k\,|\,y_{1:k-1}))\right]}_{\text{grad. w.r.t. item placement}}\underbrace{\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)}_{\text{following reward}}\Bigg],\end{aligned} \tag{13}$$

note that we use following: $\sum_{k=1}^{K}\mathbb{E}_{y_{1:k}}[f(y_{1:k})] = \mathbb{E}_y\left[\sum_{k=1}^{K}f(y_{1:k})\right]$, to move the expectation from partial rankings to complete rankings. Eq. 13 shows us that the derivative consists of two parts: the gradient w.r.t. individual item placements and the reward received following each placement. Again, this gradient can be estimated using rankings sampled from π:

$$\frac{\delta}{\delta m}\mathcal{R}(q) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left[\frac{\delta}{\delta m}\log(\pi(y_k^{(i)}|y_{1:k-1}^{(i)}))\right]\sum_{x=k}^{K}\theta_x\rho_{y_x^{(i)}}. \tag{14}$$

We will call this estimator the *placement policy gradient estimator*, in contrast with the basic policy gradient estimator (Eq. 10), this estimator weights the gradients of placement probabilities with the observed following rewards. By doing so, it makes use of the structure of ranking metrics and thus is more tailored towards these metrics than the basic estimator.

5.2 Computationally Efficient Estimation

So far our placement policy gradient estimator has made use of the fact that the probability of a ranking is a product of individual placement probabilities, however, it has made no further use of the fact that π is a PL ranking model. We will now show that using the knowledge that π is a PL model can lead to an estimator that can be computed with greater computational efficiency. We start by taking the derivative of an item placement probability:

$$\begin{aligned}&\frac{\delta}{\delta m}\pi(d\,|\,y_{1:k-1}) = \\ &\quad\pi(d\,|\,y_{1:k-1})\left(\left[\frac{\delta}{\delta m}m(d)\right] - \sum_{d'\in\mathcal{D}}\pi(d'\,|\,y_{1:k-1})\left[\frac{\delta}{\delta m}m(d')\right]\right).\end{aligned} \tag{15}$$

We note that the probability of placing an item that has already been placed is zero: $d\in y_{1:k-1}\rightarrow\pi(d\,|\,y_{1:k-1})=0$. Combining Eq. 13 & 15 results in the following gradient:

$$\begin{aligned}\frac{\delta}{\delta m}\mathcal{R}(q) &= \mathbb{E}_y\left[\sum_{k=1}^{K}\left[\frac{\delta}{\delta m}\log(\pi(y_k\,|\,y_{1:k-1}))\right]\sum_{x=k}^{K}\theta_x\rho_{y_x}\right] \\ &= \mathbb{E}_y\left[\left(\sum_{k=1}^{K}\left[\frac{\delta}{\delta m}m(y_k)\right]\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right)\right. \\ &\quad\left.-\left(\sum_{k=1}^{K}\sum_{d'\in\mathcal{D}}\pi(d'\,|\,y_{1:k-1})\left[\frac{\delta}{\delta m}m(d')\right]\right)\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right].\end{aligned} \tag{16}$$

For the sake of simplicity, we will further derive the resulting two parts of Eq. 16 separately, starting with the first part:

$$\begin{aligned}&\mathbb{E}_y\left[\left(\sum_{k=1}^{K}\left[\frac{\delta}{\delta m}m(y_k)\right]\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right)\right] \\ &= \mathbb{E}_y\left[\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\left(\sum_{k=1}^{K}\mathbb{1}[y_k=d]\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right)\right] \\ &= \mathbb{E}_y\left[\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\left(\sum_{k=1}^{K}\mathbb{1}[d\in y_{1:k}]\theta_k\rho_{y_k}\right)\right] \\ &= \sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\mathbb{E}_y\left[\sum_{k=\text{rank}(d,y)}^{K}\theta_k\rho_{y_k}\right].\end{aligned} \tag{17}$$

We see that this first part results in summing over the derivatives of each item score according to model $m(d)$ weighted by the reward expected to follow a placement of d.

Then for the second part of Eq. 16:

$$\begin{aligned}
&\mathbb{E}_y\left[\sum_{k=1}^{K}\sum_{d\in\mathcal{D}}\pi(d\,|\,y_{1:k-1})\left[\frac{\delta}{\delta m}m(d)\right]\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right]\\
&=\mathbb{E}_y\left[\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\sum_{k=1}^{K}\pi(d\,|\,y_{1:k-1})\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right]\\
&=\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\mathbb{E}_y\left[\sum_{k=1}^{\text{rank}(d,y)}\pi(d\,|\,y_{1:k-1})\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right],
\end{aligned}\tag{18}$$

where we used the fact that: $k>\text{rank}(d,y)\rightarrow\pi(d\,|\,y_{1:k-1})=0$. We see that the second part sums over each rank where it multiplies the expected probability that an item was added with the expected following reward. This product represents the *risk* imposed by an item d: if d is not placed at k then $\pi(d\,|\,y_{1:k-1})$ indicates how likely d would have been placed instead of y_k and in which case the following reward $\sum_{x=k}^{K}\theta_x\rho_{y_x}$ may not have occurred. For cases where d is the item at rank k: $d=y_k$, the risk stops the log score $m(d)$ from increasing too far as the placement probability $\pi(d\,|\,y_{1:k-1})$ may already be very great. By combining Eq. 16, 17 & 18 we obtain the full derivative:

$$\frac{\delta}{\delta m}\mathcal{R}(q)=\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\mathbb{E}_y\Bigg[\overbrace{\left(\sum_{k=\text{rank}(d,y)}^{K}\theta_k\rho_{y_k}\right)}^{\text{reward following placement}}\underbrace{-\sum_{k=1}^{\text{rank}(d,y)}\pi(d\,|\,y_{1:k-1})\left(\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)}_{\text{risk imposed by placement probability}}\Bigg].\tag{19}$$

We see that the derivative multiplies the gradient of the item log score $m(d)$ with the expected reward following its placement minus the expected risk imposed by d before it is placed. Finally, this gradient can also be estimated using N sampled rankings:

$$\begin{aligned}
\frac{\delta}{\delta m}\mathcal{R}(q)\approx\frac{1}{N}\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\sum_{i=1}^{N}\Bigg(&\sum_{k=\text{rank}(d,y^{(i)})}^{K}\theta_k\rho_{y_k^{(i)}}\Bigg)\\
&-\sum_{k=1}^{\text{rank}(d,y^{(i)})}\pi(d\,|\,y^{(i)}_{1:k-1})\left(\sum_{x=k}^{K}\theta_x\rho_{y_x^{(i)}}\right).
\end{aligned}\tag{20}$$

We call this estimator *PL-Rank-1*, to the best of our knowledge this is the first gradient estimation method that is specifically designed for optimizing PL-ranking models w.r.t. ranking metrics. While both the placement policy gradient estimator (Eq. 14) and PL-Rank-1 (Eq. 20) estimate the same gradient, their formulas look radically different. A big advantage of PL-Rank-1 is that it can be computed with a time-complexity of $O(N\cdot K\cdot D)$. Our experimental results indicate that while both estimators have comparable sample-efficiency, PL-Rank-1 requires considerably less time to compute than using a machine learning framework to automatically compute the placement policy gradient.

5.3 Improving Sample-Efficiency

In Eq. 19 we see that an item receives a positive weight from the expected following reward. Therefore, even when an item has a low probability of being placed it can compensate with a high relevance (ρ_d) to get a positive weight. However, when an estimate of the gradient is based on a low number of samples (N), they may not include a ranking where such an item is placed at all and thus these items will nevertheless receive a negative weight in the estimate. We propose one last estimator to mitigate this potential issue.

First, we can rewrite the expected reward following placement so that the reward obtained from d and that from items placed afterwards are separated:

$$\begin{aligned}
&\mathbb{E}_y\left[\sum_{k=\text{rank}(d,y)}^{K}\theta_k\rho_{y_k}\right]\\
&=\mathbb{E}_y\left[\left(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\right)+\theta_{\text{rank}(d,y)}\rho_d\right]\\
&=\mathbb{E}_y\left[\left(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\right)+\sum_{k=1}^{K}\pi(d\,|\,y_{1:k-1})\theta_k\rho_d\right]\\
&=\mathbb{E}_y\left[\left(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\right)+\sum_{k=1}^{\text{rank}(d,y)}\pi(d\,|\,y_{1:k-1})\theta_k\rho_d\right],
\end{aligned}\tag{21}$$

where again we make use of the fact that: $k>\text{rank}(d,y)\rightarrow\pi(d\,|\,y_{1:k-1})=0$. Combining this result with Eq. 19 we get:

$$\frac{\delta}{\delta m}\mathcal{R}(q)=\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\mathbb{E}_y\Bigg[\overbrace{\left(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\right)}^{\text{future reward after placement}}\underbrace{+\sum_{k=1}^{\text{rank}(d,y)}\pi(d\,|\,y_{1:k-1})\left(\theta_k\rho_d-\sum_{x=k}^{K}\theta_x\rho_{y_x}\right)}_{\text{expected direct reward minus the risk of placement}}\Bigg].\tag{22}$$

We see that the gradient w.r.t. an item's log score $m(d)$ is weighted by the reward after placement (not including the reward from d) plus the expected direct reward (the reward from d) minus the expected risk imposed by d before its placement. From Eq. 22 we can derive the following novel estimator:

$$\begin{aligned}
\frac{\delta}{\delta m}\mathcal{R}(q)\approx\frac{1}{N}\sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\sum_{i=1}^{N}\Bigg(&\sum_{k=\text{rank}(d,y^{(i)})+1}^{K}\theta_k\rho_{y_k^{(i)}}\Bigg)\\
&+\sum_{k=1}^{\text{rank}(d,y^{(i)})}\pi(d\,|\,y^{(i)}_{1:k-1})\left(\theta_k\rho_d-\sum_{x=k}^{K}\theta_x\rho_{y_x^{(i)}}\right).
\end{aligned}\tag{23}$$

We will call this estimator: *PL-Rank-2*. Unlike PL-Rank-1 (Eq. 20), PL-Rank-2 can provide a positive weight to items that were not in the top-K of any of the N sampled rankings. While this is expected to increase the sample-efficiency, it does not come at the cost of computational complexity as both PL-Rank-1 and PL-Rank-2 have a complexity of $O(N\cdot K\cdot D)$.

5.4 The PL-Rank Algorithm

Finally, we will show how PL-Rank-2 can be implemented efficiently. Our goal is to compute a λ_d weight per item d so that the gradient is estimated by:

$$\frac{\delta}{\delta m}\mathcal{R}(q) \approx \frac{1}{N}\sum_{d\in\mathcal{D}}\lambda_d\left[\frac{\delta}{\delta m}m(d)\right], \tag{24}$$

where following Eq. 23 these weights are:

$$\lambda_d = \frac{1}{N}\sum_{i=1}^{N}\left(\left(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\right) + \sum_{k=1}^{\text{rank}(d,y)}\pi(d\,|\,y_{1:k-1})\left(\theta_k\rho_d - \sum_{x=k}^{K}\theta_x\rho_{y_x}\right)\right). \tag{25}$$

Algorithm 1 displays the PL-Rank-2 algorithm in pseudo-code. As input it requires the item collection $\mathcal{D}$, the relevances ρ, the (pre-computed) log scores per item $m(d)$ according to the current model m, the metric weights per rank θ, and finally, the number of rankings to sample N (Line 1). First, N rankings are sampled using Gumbel sampling (Line 2) and zero weights are initialized for every λ (Line 2). Subsequently, the initial denominator for the PL model is computed and stored (Line 4). Then the algorithm starts iterating over each of the sampled rankings, where first, the rewards following each rank k are precomputed (Line 9). Second, it loops over every rank k where it adds the following reward to the λ_d of d at rank k in the sampled ranking (Line 11), thus computing the first part of Eq. 23. For every item, the placement probability $\pi(d\,|\,y_k^{(i)})$ is computed (Line 14) and multiplied by the difference between the item's direct reward and the following reward (Line 15), this is added to λ_d to compute the second part of Eq. 23. Finally, the denominator is updated to account for the item placed at rank k (Line 16).

Algorithm 1 reveals that PL-Rank-2 can be computed in $O(N\cdot K\cdot D)$, we note that with small alterations to Line 11 and 15 PL-Rank-1 can be computed with this algorithm as well.

Algorithm 1 PL-Rank-2 Gradient Estimation

1: **Input**: items: $\mathcal{D}$; Relevances: ρ; Metric weights: θ; Scores: m; Number of samples: N.
2: $\{y^{(1)},y^{(2)},...,y^{(N)}\} \leftarrow$ Gumbel_Sample(N,m)
3: $\lambda \leftarrow \mathbf{0}$ *// initialize zero weight per item*
4: $M \leftarrow \sum_{d\in\mathcal{D}}\exp(m(d))$ *// initialize PL denominator*
5: **for** $i \in [1,2,...,N]$ **do**
6: $\quad M' \leftarrow M$ *// copy initial PL denominator*
7: $\quad \omega_K \leftarrow \theta_K\rho_{y_K^{(i)}}$ *// reward for last rank*
8: $\quad$ **for** $k \in [K-1,K-2,...,1]$ **do**
9: $\quad\quad \omega_k \leftarrow \omega_{k+1}+\theta_k\rho_{y_k^{(i)}}$ *// pre-compute reward following rank k*
10: $\quad$ **for** $k \in [1,2,...,K]$ **do**
11: $\quad\quad \lambda_{y_k^{(i)}} \leftarrow \lambda_{y_k^{(i)}}+\omega_{k+1}$ *// add future reward k*
12: $\quad\quad$ **for** $d \in \mathcal{D}$ **do**
13: $\quad\quad\quad$ **if** $d \notin y_{1:k-1}^{(i)}$ **then**
14: $\quad\quad\quad\quad P \leftarrow (\exp(m(d))/M')$ *// placement probability*
15: $\quad\quad\quad\quad \lambda_d \leftarrow \lambda_d + P\cdot(\theta_k\rho_d-\omega_k)$ *// 2nd part of Eq. 23*
16: $\quad\quad M' \leftarrow M' - \exp(m(y_k^{(i)}))$ *// renormalize denominator*
17: **return** $\frac{1}{N}\lambda$

6 METHOD: PL-RANK FOR FAIRNESS

So far we have introduced the PL-Rank algorithms for estimating the gradient of a PL ranking model w.r.t. a relevance metric. However, the applicability of these algorithms are much wider than just relevance metrics, in particular, they can be applied to any *exposure-based* metrics [2, 11, 20, 28]. Exposure represents the expected number of people that will examine an item. In general, user behavior has *position-bias* which means that they are less likely to examine an item if it displayed at a lower rank [9, 30]. Let the rank weight θ_k indicate the probability that a user examines an item at rank k, then the exposure an item d receives under π is:

$$\mathcal{E}(q,d) = \mathbb{E}_y\left[\sum_{k=1}^{K}\theta_k\mathbb{1}[y_k=d]\right] = \sum_{y\in\pi}\pi(y)\sum_{k=1}^{K}\theta_k\mathbb{1}[y_k=d], \tag{26}$$

where again for brevity we denote $\mathcal{E}_d = \mathcal{E}(q,d)$. Thus $\mathcal{E}_d$ could be interpreted as the probability that a user examines d when π is deployed. Most fairness metrics for rankings consider how exposure is distributed over items and specifically how *fair* this distribution is. Regardless of the exact metric, PL-Rank can be applied to a fairness metric $\mathcal{F}$ if the chain-rule can be applied as follows:

$$\frac{\delta}{\delta m}\mathcal{F}(q) = \sum_{d\in\mathcal{D}}\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_d}\frac{\delta\mathcal{E}_d}{\delta m}. \tag{27}$$

To derive this gradient, we first note that the $\mathcal{E}_d$ (Eq. 26) and $\mathcal{R}(q)$ (Eq. 1) are equivalent if $\forall d'\,\rho_{d'} = \mathbb{1}[d'=d]$, therefore if we replace ρ in PL-Rank-2 (Eq. 22) accordingly, it will provide us the gradient $\frac{\delta\mathcal{E}_d}{\delta m}$. If we combine this fact with Eq. 27 we obtain the following PL-Rank-2 based gradient:

$$\frac{\delta}{\delta m}\mathcal{F}(q) = \sum_{d\in\mathcal{D}}\left[\frac{\delta}{\delta m}m(d)\right]\mathbb{E}_y\left[\left(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\left[\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_{y_k}}\right]\right) + \sum_{k=1}^{\text{rank}(d,y)}\pi(d|y_{1:k-1})\left(\theta_k\left[\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_d}\right] - \sum_{x=k}^{K}\theta_x\left[\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_{y_x}}\right]\right)\right]. \tag{28}$$

In other words, we can apply PL-Rank by simply replacing the item relevances with the gradients: $\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_d}$ before computation. Similarly, any linear combination of $\mathcal{R}$ and $\mathcal{F}$ can be optimized by replacing the relevances with the corresponding linear combination between ρ_d and $\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_d}$.

For instance, we can follow Singh and Joachims [28] and choose a disparity-based metric. This metric measures the disparity between two items via a function $D(d,d')$ and takes the average disparity over all item pairs:

$$\mathcal{F}(q) = \frac{1}{|\mathcal{D}|(|\mathcal{D}|-1)}\sum_{d\in\mathcal{D}}\sum_{d'\in\mathcal{D}}D(d,d'). \tag{29}$$

Singh and Joachims [28] divide the exposure of an item d by its relevance: $\frac{\mathcal{E}_d}{\rho_d}$ to model the proportion between the exposure and the merit of an item. However, in our experimental datasets many items have zero relevances, thus making such a division impossible. Instead, we introduce a novel alternative disparity measure:

$$D(d,d') = (\mathcal{E}_{d'}\rho_d - \mathcal{E}_d\rho_{d'})^2. \tag{30}$$

This measure looks at the reward item d would receive if it had the exposure of d': $\mathcal{E}_{d'}\rho_d$, in other words, the reward d would receive if it was treated as d' is. This measure can handle items without merit and has the gradient:

$$\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_d} = \frac{4}{|\mathcal{D}|(|\mathcal{D}|-1)} \sum_{d'\in\mathcal{D}} (\mathcal{E}_{d'}\rho_d - \mathcal{E}_d\rho_{d'})\rho_{d'}. \tag{31}$$

Thus in order to apply PL-Rank-2 to this fairness metric, one only needs to compute (or estimate) Eq. 31 and then run Algorithm 1 where the relevances are replaced with the gradients: $\frac{\delta\mathcal{F}(q)}{\delta\mathcal{E}_d}$.

To conclude, we have shown that PL-Rank-2 can efficiently estimate the gradient of exposure-based fairness metrics, in addition to relevance metrics and any linear combination of any set of these metrics.

7 EXPERIMENTAL SETUP

The experiments performed for this paper aim to answer three research questions:

RQ1 Does PL-Rank require fewer sampled rankings for optimal convergence than policy gradients or LambdaLoss?

RQ2 Is less computational time needed to reach high performance with PL-Rank than with policy gradients or LambdaLoss?

RQ3 Is PL-Rank also effective at optimizing an exposure-based fairness metric?

In other words, we address the sample-efficiency and the computational costs of PL-Rank, in addition to its applicability to ranking-fairness metrics.

To evaluate these aspects we compare with three baselines: (i) the policy gradient as described in Section 4.2, this is the most basic form of gradient estimation [3, 28, 33]; (ii) the placement policy gradient as introduced in Section 5.1, this gradient estimation considers individual item placements; and (iii) LambdaLoss [31], a state-of-the-art heuristic for optimizing deterministic ranking models. Following Bruch et al. [3] we apply an average of the gradients over N sampled rankings. One can easily extend the existing proof that LambdaLoss optimizes a lower bound on the performance of a deterministic model [31] to prove our approach also optimizes a lower bound on the expected performance of a stochastic PL ranking model. We note that Bruch et al. [3] introduced additional heuristic methods for PL-Ranking model optimization, due to their high similarity with LambdaLoss we omitted these methods from our baselines. To the best of our knowledge, our choice of baselines cover every category of existing methods for the metric-based optimization of PL-Ranking models.

We base our experiments on the three largest publicly-available LTR industry datasets: *Yahoo! Webscope* [8], *MSLR-WEB30k* [26], and *Istella* [10]. Each dataset contains queries, preselected documents per query, and relevance labels indicating the expert-judged relevance of a preselected document w.r.t. a query. Query-document combinations are represented by feature vectors, each dataset varies in the number of features, queries and average number of preselected documents: Yahoo contains 29,921 queries and on average 24 preselected documents per query encoded in 700 features; MSLR has 30,000 queries, on average 125 documents per query and 136 features; and lastly, Istella has 33,118 queries, on average 315 documents per query and 220 features.

Table 1: Average time in minutes taken to perform one training epoch for different numbers of sampled rankings N, the standard deviation is displayed in brackets.

		$N=1$	$N=10$	$N=100$	$N=1000$
Yahoo	LambdaLoss	2.48 (0.05)	2.53 (0.04)	3.06 (0.08)	10.25 (0.53)
	Policy Gradient	3.79 (0.09)	3.80 (0.06)	4.28 (0.15)	8.27 (0.50)
	Placement P.G.	3.83 (0.08)	3.86 (0.05)	4.42 (0.10)	8.26 (0.44)
	PL-Rank-1	**2.45 (0.06)**	**2.49 (0.06)**	**2.82 (0.09)**	**5.70 (0.14)**
	PL-Rank-2	2.49 (0.06)	2.52 (0.06)	2.87 (0.08)	6.22 (0.15)
MSLR	LambdaLoss	2.73 (0.11)	3.96 (0.59)	36.36 (31.46)	1669.59 (450.69)
	Policy Gradient	3.30 (0.10)	3.45 (0.10)	5.25 (0.35)	24.20 (2.77)
	Placement P.G.	3.32 (0.17)	3.42 (0.13)	5.27 (0.41)	23.97 (2.68)
	PL-Rank-1	**2.16 (0.14)**	**2.28 (0.13)**	**3.34 (0.13)**	**18.48 (2.10)**
	PL-Rank-2	2.19 (0.15)	2.35 (0.17)	3.45 (0.06)	21.10 (2.78)
Istella	LambdaLoss	3.53 (0.12)	4.50 (0.10)	27.81 (19.20)	142.74 (16.25)
	Policy Gradient	4.10 (0.17)	4.51 (0.16)	8.74 (0.26)	44.29 (2.17)
	Placement P.G.	4.08 (0.17)	4.51 (0.18)	8.72 (0.23)	44.74 (2.55)
	PL-Rank-1	**3.01 (0.12)**	**3.27 (0.09)**	**6.90 (0.19)**	**39.80 (2.65)**
	PL-Rank-2	3.04 (0.13)	3.31 (0.10)	7.02 (0.14)	40.64 (3.96)

For our relevance experiments, we optimize top-5 *Discounted Cumulative Gain* (DCG@5) and choose θ accordingly (Eq. 3).[1] The relevance of a document is set to a transformation of its label: $\rho_d = 2^{\text{relevance\_label}(d)} - 1$. For the fairness experiments, we optimize the disparity metric introduced in Section 6, exposure values $\mathcal{E}_d$ are estimated using 1000 sampled rankings. To compare the computational costs of each method, we ran repeated experiments under identical circumstances on a single *Intel Xeon Silver 4214* CPU and measured the time taken to complete each epoch. All our reported results are averaged over 20 independent runs.

Based on preliminary parameter tuning, we chose to optimize neural networks with two hidden layers of 32 sigmoid activated nodes, we used standard stochastic gradient descent with a 0.01 learning rate for all methods. For calculating gradients we utilize *Tensorflow* [1] with two exceptions: the sampling of rankings and $\frac{\delta\mathcal{R}(q)}{\delta m}$ with the PL-Rank algorithm (Algorithm 1) are computed using *Numpy* [15].

8 RESULTS

Our discussion of the results is divided per research question, our results are displayed in Figure 1, 2 and 3 and Table 1 and 2.

8.1 Sample-Efficiency

We will first consider **RQ1**: *whether PL-Rank needs fewer sampled rankings for optimal convergence.* Figure 1 shows the performance of PL ranking models trained using different gradient estimation methods with varying numbers of sampled rankings used for estimation. We see that increasing the number of samples beyond $N=10$ does not have any noticeable effect on the performance of LambdaLoss. In all cases, LambdaLoss converges at suboptimal performance after only a few epochs. In contrast, the basic policy gradient is very affected by N and on all three datasets it requires $N=1000$ to get close to optimal performance, it has extreme convergence issues when $N=10$. However, in all cases the placement policy gradient

[1] We chose DCG instead of normalized DCG based on the advice of Ferrante et al. [13].

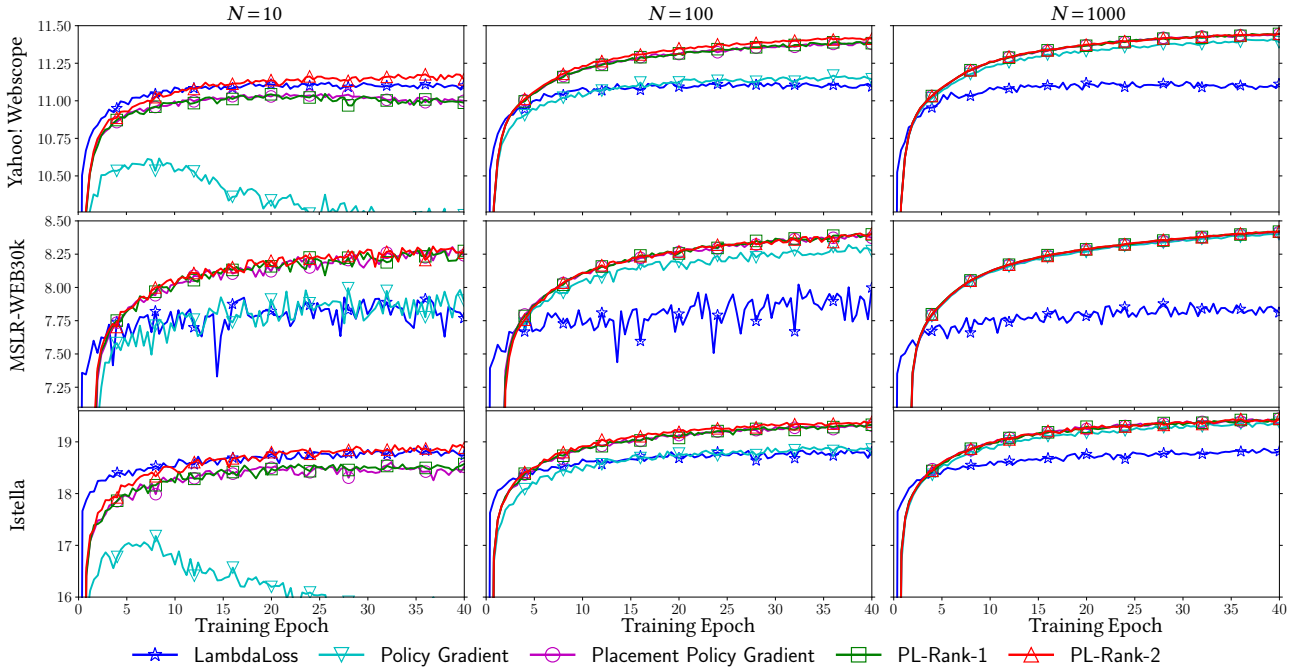

Figure 1: Performance in DCG@5 of PL ranking models trained using different gradient estimation methods with varying number of sampled rankings N per estimation. Results are the mean of 20 independent runs.

outperforms the basic policy gradient and can converge near optimal performance with $N = 100$. The performance of PL-Rank-1 and the placement policy gradient are indistinguishable. This strongly suggests that PL-Rank-1 and the placement policy gradient perform the same estimation, although Section 8.2 will reveal that PL-Rank-1 does so in a more computationally efficient way. Lastly, we see that PL-Rank-2 outperforms PL-Rank-1 and the policy gradient methods when $N = 10$ on the Yahoo and Istella datasets. Noticeable but limited improvements are also present with $N = 100$ on these datasets. Thus while we can conclude that PL-Rank-2 has the best sample-efficiency of all the methods, the improvements over PL-Rank-1 and the placement policy gradient are most substantial when $N < 100$.

Overall, we see that the basic policy gradient and LambdaLoss are poor choices for optimization, despite the fact that these are the methods we find in previous work [3, 28]. The choice between PL-Rank-1 and the placement policy gradient does not seem to matter when only the number of epochs is considered. PL-Rank-2 appears the safest choice because, in all tested cases, it either outperforms or has comparable performance to the other methods.

To conclude, we answer **RQ1** in the affirmative: PL-Rank-2 is the most sample-efficient method, although when $N > 100$ PL-Rank-1 and the placement policy gradient have comparable performance.

8.2 Computational Costs and Time-Efficiency

In order to answer **RQ2**: *whether PL-Rank requires less computational time to reach optimal performance*, we first consider Table 1 which displays the average time taken to perform a single epoch per method for various N values.

We see that in all cases PL-Rank-1 takes the least amount of time to compute, with PL-Rank-2 being the second fastest method. There is little difference between the policy gradient methods but they are always much slower than the PL-Rank methods. Depending on the dataset and N, the difference between PL-Rank and the policy gradients varies from around a minute to almost five minutes. In Figure 1 we see that convergence requires at least 40 training epochs, thus differences in minutes per epoch can easily add up to reaching convergence over an hour earlier.

Table 2: DCG@5 reached using different gradient estimation methods following a dynamically updated N and being optimized for the same amount of time. Results are the mean of 20 independent runs, the standard deviation is displayed in brackets, $\triangledown$ indicates the result is significantly worse ($p < 0.01$) than that of PL-Rank-2 on the same dataset.

	Yahoo	MSLR	Istella
Minutes Optimized	100	120	200
LambdaLoss	11.11 $\triangledown$ (0.05)	7.80 $\triangledown$ (0.09)	18.75 $\triangledown$ (0.10)
Policy Gradient	11.03 $\triangledown$ (0.04)	8.21 $\triangledown$ (0.07)	18.75 $\triangledown$ (0.10)
Placement Policy Gradient	11.31 $\triangledown$ (0.02)	8.33 $\triangledown$ (0.04)	19.23 $\triangledown$ (0.06)
PL-Rank-1	11.38 $\triangledown$ (0.03)	**8.39** – **(0.04)**	19.31 $\triangledown$ (0.05)
PL-Rank-2	**11.42** – **(0.02)**	**8.39** – **(0.03)**	**19.38** – **(0.05)**

LambdaLoss is especially affected by the N parameter, a likely explanation is that it is the only method that has to sample the complete ranking, whereas the other methods only need to sample the top-k ranking (top-5 in this case).

By considering both the results from Table 1 and Figure 1, we can make three observations: (i) when $N = 10$, decent but not optimal performance is reached; (ii) $N = 100$ is enough to converge near optimal performance; and (iii) performing an epoch with $N = 10$ is considerably faster than with $N = 100$. Based on these observations, it seems reasonable to increase N at every training step so that decent performance is reached very quickly but convergence is still optimal.

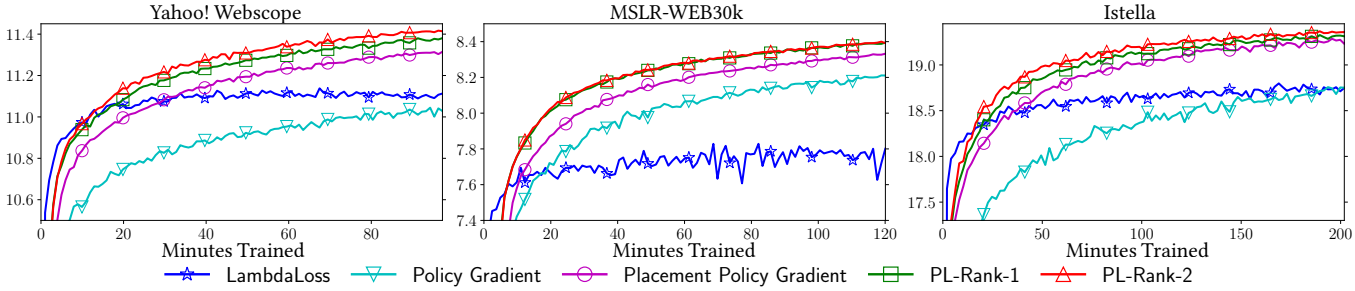

Figure 2: Performance in DCG@5 of PL ranking models trained using different gradient estimation methods following a dynamically updated number of sampled rankings N per estimation. Results are the mean of 20 independent runs.

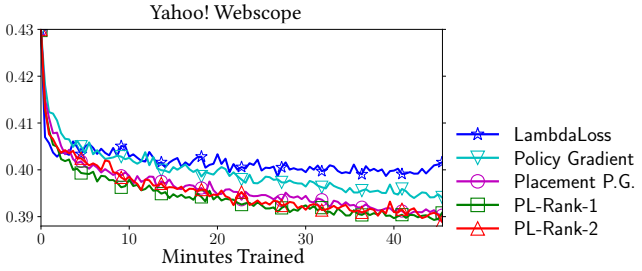

Figure 3: The mean disparity error of models trained using different gradient estimation methods following a dynamically updated number of sampled rankings N per estimation. Results are the mean of 20 independent runs.

We found that $N = 10 + 90 \cdot \frac{\text{epoch}}{40}$ outperformed the static choices $N = 10$ and $N = 100$ in terms of learning speed while maintaining optimal convergence.

Figure 2 shows the performance of PL ranking models trained using this dynamic N strategy over training time in minutes. Again we see that LambdaLoss converges fast but at suboptimal performance. Similarly, there is clearly a large difference visible between the basic policy gradient and the placement policy gradient. However, on all datasets, we see an improvement of PL-Rank-1 over the placement policy gradient which is very large on Yahoo and MSLR but smaller on Istella. This improvement can be attributed to the reduced computational costs of PL-Rank-1, as a result, it is capable of completing more epochs in the same amount of time and can therefore reach a higher performance in less computational time. Finally, compared to PL-Rank-1, PL-Rank-2 has an even higher performance on the Yahoo and Istella datasets but not on MSLR. It appears that its increased sample-efficiency helps PL-Rank-2 initially, when N is low, except on the MSLR dataset where Figure 1 also shows us that the difference in sample-efficiency is very limited. To better verify that PL-Rank-2 is the best choice, we performed a two-sided student t-test on the performance differences, the results are displayed in Table 2. We see that Pl-Rank-2 is significantly better compared to the other methods in all tested cases, with the single exception of PL-Rank-1 on the MSLR dataset.

To conclude, we answer **RQ2** in the affirmative: the PL-Rank methods achieves significantly higher performance with less computational time required than LambdaLoss or the policy gradients. In particular, PL-Rank-2 is the most time-efficient method across all datasets.

8.3 Optimizing a Ranking Fairness Metric

Finally, we address **RQ3**: *whether PL-Rank is effective at optimizing ranking fairness.* Figure 3 displays the mean disparity error for models optimized with the different gradient estimation methods, with the same increasing N strategy applied as in Section 8.2. While all methods decrease the disparity, the PL-Rank methods and the placement policy gradient are considerably more efficient than LambdaLoss and the basic policy gradient. Unlike with the optimization for relevance, there appears only a very small improvement of the PL-Rank methods over the placement policy gradient. Therefore, we answer **RQ3** in the affirmative: PL-Rank can effectively optimize ranking fairness, where we note that the placement policy gradient has comparable time-efficiency.

9 CONCLUSION

In this paper, we tackled the optimization of PL-ranking models for both relevance and fairness ranking metrics. While previous work has found PL-ranking models effective for various ranking tasks, their optimization can involve large computational costs. To alleviate these costs, we introduced three new estimators for efficiently estimating the gradient of a ranking metric w.r.t. a PL ranking model: the placement policy gradient and two PL-Rank methods. The latter two can be computed using the PL-Rank algorithm. To the best of our knowledge, PL-Rank is the first algorithm designed specifically for efficiently optimizing PL ranking models w.r.t. ranking metrics. Our experimental results indicate that our novel methods considerably reduce the computational time required to reach optimal performance compared to existing methods. In particular, the PL-Rank-2 method has the best sample-effiency and was found to reach significantly higher performance when ran for the same amount of time as other methods. Compared to the popular basic policy gradient, PL-Rank-2 converges several hours earlier, thus immensely alleviating the computational costs of optimization.

With the introduction of PL-Rank, we hope that the usage of stochastic ranking models is made more attractive in real-world scenarios. Finally, we think PL-Rank is also an important theoretical contribution to the LTR field, as it proves that PL ranking models can be optimized with computational efficiency, without relying on heuristic methods.

Code and data

To facilitate reproducibility, this work only made use of publicly available data and our experimental implementation is publicly available at https://github.com/HarrieO/2021-SIGIR-plackett-luce.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation OSDI'16)*. 265–283.

[2] Asia J Biega, Krishna P Gummadi, and Gerhard Weikum. 2018. Equity of attention: Amortizing individual fairness in rankings. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 405–414.

[3] Sebastian Bruch, Shuguang Han, Michael Bendersky, and Marc Najork. 2020. A Stochastic Treatment of Learning to Rank Scoring Functions. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 61–69.

[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*. 1241–1244.

[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*. 89–96.

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

[7] 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*. 129–136.

[8] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research* 14 (2011), 1–24.

[9] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*. 87–94.

[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 Transactions on Information Systems (TOIS)* 35, 2 (2016), Article 15.

[11] Fernando Diaz, Bhaskar Mitra, Michael D. Ekstrand, Asia J. Biega, and Ben Carterette. 2020. *Evaluating Stochastic Rankings with Expected Exposure*. Association for Computing Machinery, New York, NY, USA, 275–284.

[12] Pinar Donmez, Krysta M Svore, and Christopher JC Burges. 2009. On the local optimality of LambdaRank. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 460–467.

[13] Marco Ferrante, Nicola Ferro, and Norbert Fuhr. 2021. Towards Meaningful Statements in IR Evaluation. Mapping Evaluation Measures to Interval Scales. *arXiv preprint arXiv:2101.02668* (2021).

[14] Emil Julius Gumbel. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office.

[15] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'10, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (2020), 357–362.

[16] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2011. A Probabilistic Method for Inferring Preferences from Clicks. In *CIKM*. ACM, 249–258.

[17] Thorsten Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 133–142.

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

[19] R Duncan Luce. 2012. *Individual choice behavior: A theoretical analysis*. Courier Corporation.

[20] Marco Morik, Ashudeep Singh, Jessica Hong, and Thorsten Joachims. 2020. *Controlling Fairness and Bias in Dynamic Learning-to-Rank*. ACM, 429–438.

[21] Harrie Oosterhuis and Maarten de Rijke. 2018. Differentiable Unbiased Online Learning to Rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 1293–1302.

[22] Harrie Oosterhuis and Maarten de Rijke. 2020. Taking the Counterfactual Online: Efficient and Unbiased Online Evaluation for Ranking. In *Proceedings of the 2020 International Conference on The Theory of Information Retrieval*. ACM.

[23] Harrie Oosterhuis and Maarten de Rijke. 2021. Unifying Online and Counterfactual Learning to Rank. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM'21)*. ACM.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.

[25] Robin L Plackett. 1975. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 24, 2 (1975), 193–202.

[26] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597* (2013).

[27] Anne Schuth, Robert-Jan Bruintjes, Fritjof Buüttner, Joost van Doorn, Carla Groenland, Harrie Oosterhuis, Cong-Nguyen Tran, Bas Veeling, Jos van der Velde, Roger Wechsler, et al. 2015. Probabilistic multileave for online retrieval evaluation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 955–958.

[28] Ashudeep Singh and Thorsten Joachims. 2019. Policy learning for fairness in ranking. In *Advances in Neural Information Processing Systems*. 5426–5436.

[29] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. 77–86.

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

[31] 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*. ACM, 1313–1322.

[32] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2017. Reinforcement learning to rank with Markov decision process. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 945–948.

[33] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[34] 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*. 1192–1199.

[35] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, Wei Zeng, and Xueqi Cheng. 2017. Adapting Markov decision process for search result diversification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 535–544.