Computationally Efficient Optimization of Plackett-Luce Ranking Models for Relevance and Fairness (Extended Abstract)

Harrie Oosterhuis
Radboud University
harrie.oosterhuis@ru.nl

Abstract

Computing the gradient of stochastic Plackett-Luce (PL) ranking models for relevance and fairness metrics can be infeasible because it requires iterating over all possible permutations of items.

In this paper, we introduce a novel algorithm: PL-Rank, that estimates the gradient of a PL ranking model through sampling. Unlike existing approaches, 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.

1 Introduction

Learning to Rank (LTR) is a branch of machine learning that covers methods for optimizing ranking systems for search and recommendation applications [Liu, 2009]. Traditionally, scoring function are used to assigns an individual score to each item, and subsequently, produce rankings by sorting items according to their assigned scores [Fuhr, 1989; Joachims, 2002]. The main difficulty in LTR is that there is no gradient w.r.t. the sorting procedure. Solutions to this difficulty can be divided into two categories: optimizing a heuristic function that bounds or approximates the ranking performance [Burges, 2010; Wang *et al.*, 2018; Bruch *et al.*, 2019]; or optimizing a probabilistic ranking system [Cao *et al.*, 2007; Xia *et al.*, 2008; Ustimenko and Prokhorenkova, 2020].

In recent years, the popularity of the Plackett-Luce (PL) ranking model has increased [Singh and Joachims, 2019; Diaz *et al.*, 2020]. It models ranking as a succession of decision problems where each individual decision is made by a PL model [Plackett, 1975; Luce, 2012]. Previous research from the industry indicates that the probabilistic nature of the PL model leads to more robust performance [Bruch *et al.*, 2020]. In online LTR, it appears the PL model is very good at exploration because it explicitly quantifies its uncertainty [Oosterhuis and de Rijke, 2018]. Recent work has also posed that the PL model is well suited to address fairness aspects of ranking [Singh and Joachims, 2019; Diaz *et al.*, 2020].

However, computing the gradient of a PL ranking model requires an iteration over every possible permutation of items. In practice, this computational infeasibility can be circumvented by estimating the gradient based on rankings sampled from the model [Oosterhuis and de Rijke, 2020]. Unfortunately, this approach can be very computationally costly.

In this paper, we introduce PL-Rank: a novel method that can efficiently optimize both relevance and exposure-based fairness ranking metrics. Our work contributes novel estimators that unbiasedly estimate the gradient of a PL ranking model w.r.t. a ranking metric, and the PL-Rank algorithm to compute them efficiently. 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 and increased time-efficiency: PL-Rank requires less computational time to converge at optimal performance. 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 [Diaz *et al.*, 2020].

2 Background

Let $\rho_{q,d} = \rho_d$ denote the *relevance* of item d for a query q (for brevity we omit q in notation), let y be a ranking and y_k denote the kth item in ranking y: $y = [y_1, y_2, \ldots, y_K]$. Furthermore, $\pi(y \mid q) = \pi(y)$ indicates the probability that ranking y is sampled for query q by ranking model π. The performance of π for a single query is determined by the weights per rank θ and an expectation over the ranking behavior of π:

$$\mathcal{R}(q) = \sum_{y \in \pi} \pi(y) \sum_{k=1}^{K} \theta_k \rho_{y_k} = \mathbb{E}_y \left[\sum_{k=1}^{K} \theta_k \rho_{y_k} \right]. \quad (1)$$

By choosing θ_k accordingly, $\mathcal{R}(q)$ can represent the most common relevance ranking metrics: e.g. top-K Discounted Cumulative Gain (DCG): $\theta_k^{\text{DCG@K}} = \mathbb{1}[k \leq K] / \log_2(k+1)$, or precision at K: $\theta_k^{\text{PREC@K}} = \frac{1}{K} \mathbb{1}[k \leq K]$. The overall performance of a π is the expected performance over the *natural* distribution of user-issued queries:

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

Accordingly, LTR for relevance optimizes π to maximize $\mathcal{R}$, given relevances ρ_d and the chosen metric represented by θ_k.

In the PL ranking model [Bruch *et al.*, 2020], 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 $m(q,d) = m(d) \in \mathbb{R}$. The probability that an unplaced 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:

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

where $y_{1:k-1}$ indicates the ranking up to rank $k-1$, i.e., $y_{1:k-1} = [y_1, y_2, \ldots, y_{k-1}]$. We note the large similarity with the Soft-Max function commonly used in deep learning. Accordingly, the probability of a ranking is simply the product of the placement probabilities of each individual item:

$$\pi(y) = \prod_{k=1}^{K} \pi(y_k \mid y_{1:k-1}, \mathcal{D}). \tag{4}$$

An advantage of the PL ranking model is that rankings can be sampled quite efficiently, by applying the Gumbel Softmax trick [Gumbel, 1954]: for each item a sample from the Gumbel distribution is taken: $\gamma_d^{(i)} \sim \text{Gumbel}(0,1)$, i.e. 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: $\hat{m}_d^{(i)} = m(d) + \gamma_d^{(i)}$. Finally, sorting according to the $\hat{m}^{(i)}$ values provides a sampled ranking:

$$\begin{aligned} y^{(i)} &= [y_1^{(i)}, y_2^{(i)}, \ldots, y_K^{(i)}] \\ &\text{s.t.} \quad \forall (y_x^{(i)}, y_z^{(i)}), \quad x < z \rightarrow \hat{m}_{y_x}^{(i)} \geq \hat{m}_{y_z}^{(i)}. \end{aligned} \tag{5}$$

This sampling procedure follows the PL distribution of π [Gumbel, 1954; Bruch *et al.*, 2020]. In practice, rankings can thus be sampled as quickly as top-K sorting, which translates to a computational complexity of $\mathcal{O}(|\mathcal{D}| + K \log(|\mathcal{D}|))$.

Previous work have used policy-gradients to optimize PL ranking models [Singh and Joachims, 2019; Bruch *et al.*, 2020], they utilize the famous *log-trick* from the *REINFORCE* algorithm [Williams, 1992]:

$$\frac{\delta}{\delta m} \pi(y) = \pi(y) \left[\frac{\delta}{\delta m} \log(\pi(y)) \right]. \tag{6}$$

By applying the log-trick to Eq. 1, its derivate can be expressed as an expectation over the ranking distribution π:

$$\begin{aligned} \frac{\delta}{\delta m} \mathcal{R}(q) &= \sum_{y \in \pi} \left[\frac{\delta}{\delta m} \pi(y) \right] \sum_{k=1}^{K} \theta_k \rho_{y_k} \\ &= \mathbb{E}_y \Bigg[\underbrace{\left[\frac{\delta}{\delta m} \log(\pi(y)) \right]}_{\text{gradient w.r.t. complete ranking}} \quad \underbrace{\left(\sum_{k=1}^{K} \theta_k \rho_{y_k} \right)}_{\text{full reward}} \Bigg]. \end{aligned} \tag{7}$$

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. Luckily, to avoid iterating over all possible rankings the gradient can approximated by a simple sampling strategy; If we sample N rankings from π with $y^{(i)}$ denoting the ith sample, then the following estimates the gradient unbiasedly:

$$\frac{\delta}{\delta m} \mathcal{R}(q) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\delta}{\delta m} \log(\pi(y^{(i)})) \right] \left(\sum_{k=1}^{K} \theta_k \rho_{y_k^{(i)}} \right). \tag{8}$$

A straightforward implementation could use a machine learning framework to compute the gradient w.r.t. the log probabilities: $\left[\frac{\delta}{\delta m} \log(\pi(y^{(i)})) \right]$, e.g. *PyTorch* [Paszke and others, 2019] or *Tensorflow* [Abadi and others, 2016]. The downside of this approach is that it delegates the most difficult computation to a framework, which generally do not make use of the fact that π is a PL model and that a ranking metric is optimized. In contrast, the remainder of this paper will introduce methods that make use of specific PL properties, and as a result, show better computational efficiency in our experimental results.

3 Method: PL-Rank for Relevance

In this section, we will derive three novel methods for estimating the gradient of a PL-ranking model.

3.1 Ranking Metric Based Approximation

By rewriting Eq. 1 we can see that ρ_d only need to be multiplied with the probability of the ranking up to their rank:

$$\mathcal{R}(q) = \sum_{y \in \pi} \pi(y) \sum_{k=1}^{K} \theta_k \rho_{y_k} = \sum_{k=1}^{K} \theta_k \sum_{y_{1:k} \in \pi} \pi(y_{1:k}) \rho_{y_k}, \tag{9}$$

where $\sum_{y_{1:k} \in \pi}$ is a summation over all possible (sub)rankings of length k. Intuitively this makes sense because the placement of any item after k will not affect the relevance of previously placed items.

Let us 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. 4:

$$\left[\frac{\delta}{\delta m} \pi(y_{1:k}) \right] = \pi(y_{1:k}) \sum_{x=1}^{k} \left[\frac{\delta}{\delta m} \log(\pi(y_x | y_{1:x-1})) \right]. \tag{10}$$

Eq. 9 & 10 allow us to get the following derivative of $\mathcal{R}(q)$:

$$\begin{aligned} \frac{\delta}{\delta m} \mathcal{R}(q) &= \sum_{k=1}^{K} \theta_k \sum_{y_{1:k} \in \pi} \rho_{y_k} \left[\frac{\delta}{\delta m} \pi(y_{1:k}) \right] \\ &= \mathbb{E}_y \Bigg[\sum_{k=1}^{K} \underbrace{\left[\frac{\delta}{\delta m} \log(\pi(y_k \mid y_{1:k-1})) \right]}_{\text{gradient w.r.t. item placement}} \underbrace{\left(\sum_{x=k}^{K} \theta_x \rho_{y_x} \right)}_{\text{following reward}} \Bigg]. \end{aligned} \tag{11}$$

Again, this gradient can be estimated using sampled rankings:

$$\frac{\delta}{\delta m} \mathcal{R}(q) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \left[\frac{\delta}{\delta m} \log(\pi(y_k^{(i)} | y_{1:k-1}^{(i)})) \right] \sum_{x=k}^{K} \theta_x \rho_{y_x^{(i)}}. \tag{12}$$

We will call this estimator the *placement policy gradient estimator*, in contrast with the basic policy gradient estimator (Eq. 8), this estimator weights the gradients of placement probabilities with the observed following relevances. By doing so, it uses of the structure of ranking metrics and thus is more tailored towards these metrics than the basic estimator.

3.2 Computationally Efficient Estimation

Next, we will use the knowledge that π is a PL model to propose an estimator that can be computed with greater computational efficiency. We start by taking the derivative of an item placement probability:

$$\frac{\delta}{\delta m}\pi(d \mid y_{1:k-1}) = \pi(d \mid y_{1:k-1})\Big(\Big[\frac{\delta}{\delta m}m(d)\Big] - \sum_{d' \in \mathcal{D}} \pi(d' \mid y_{1:k-1})\Big[\frac{\delta}{\delta m}m(d')\Big]\Big). \tag{13}$$

We note that the probability of placing an item that has already been placed is zero: $d \in y_{1:k-1} \rightarrow \pi(d \mid y_{1:k-1}) = 0$. Combining Eq. 11 & 13 results in the following gradient:

$$\begin{aligned}\frac{\delta}{\delta m}\mathcal{R}(q) &= \mathbb{E}_y\Bigg[\sum_{k=1}^{K}\Big[\frac{\delta}{\delta m}\log(\pi(y_k \mid y_{1:k-1}))\Big]\sum_{x=k}^{K}\theta_x\rho_{y_x}\Bigg] \\ &= \sum_{d\in\mathcal{D}}\Big[\frac{\delta}{\delta m}m(d)\Big]\mathbb{E}_y\Bigg[\overbrace{\Bigg(\sum_{k=\text{rank}(d,y)}^{K}\theta_k\rho_{y_k}\Bigg)}^{\text{reward following placement}} \\ &\quad - \underbrace{\sum_{k=1}^{\text{rank}(d,y)}\pi(d \mid y_{1:k-1})\Bigg(\sum_{x=k}^{K}\theta_x\rho_{y_x}\Bigg)}_{\text{risk imposed by placement probability}}\Bigg].\end{aligned} \tag{14}$$

We see that the derivative can seen as a summation over items with two parts per item: the first part represents the expected reward to follow the placement of d; the second part can be interpreted as a the *risk* imposed by an item d. We view it as representing a risk since if d is not placed at k then $\pi(d \mid y_{1:k-1})$ 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 $\pi(d \mid y_{1:k-1})$ may already be very great.

Based on Eq. 14, we introduce the *PL-Rank-1* estimator:

$$\begin{aligned}\frac{\delta}{\delta m}\mathcal{R}(q) \approx \frac{1}{N}\sum_{d\in\mathcal{D}}\Big[\frac{\delta}{\delta m}m(d)\Big]\sum_{i=1}^{N}\Bigg(\sum_{k=\text{rank}(d,y^{(i)})}^{K}\theta_k\rho_{y_k^{(i)}}\Bigg) \\ - \sum_{k=1}^{\text{rank}(d,y^{(i)})}\pi(d \mid y^{(i)}_{1:k-1})\Bigg(\sum_{x=k}^{K}\theta_x\rho_{y_x^{(i)}}\Bigg).\end{aligned} \tag{15}$$

To the best of our knowledge, PL-Rank-1 is the first gradient estimation method that is specifically designed for optimizing PL-ranking models w.r.t. ranking metrics. Importantly, it can be computed with a time-complexity of $\mathcal{O}(N \cdot K \cdot D)$.

3.3 Improving Sample-Efficiency

In Eq. 14 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, the N samples 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 estimated gradient. We propose one last estimator to mitigate this potential issue.

First, we rewrite the expected reward following placement to separate the reward obtained from d:

$$\begin{aligned}&\mathbb{E}_y\Bigg[\sum_{k=\text{rank}(d,y)}^{K}\theta_k\rho_{y_k}\Bigg] \\ &= \mathbb{E}_y\Bigg[\Bigg(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\Bigg) + \sum_{k=1}^{\text{rank}(d,y)}\pi(d \mid y_{1:k-1})\theta_k\rho_d\Bigg].\end{aligned} \tag{16}$$

Combining this result with Eq. 14 we get:

$$\begin{aligned}\frac{\delta}{\delta m}\mathcal{R}(q) = \sum_{d\in\mathcal{D}}\Big[\frac{\delta}{\delta m}m(d)\Big]\mathbb{E}_y\Bigg[\overbrace{\Bigg(\sum_{k=\text{rank}(d,y)+1}^{K}\theta_k\rho_{y_k}\Bigg)}^{\text{future reward after placement}} \\ + \underbrace{\sum_{k=1}^{\text{rank}(d,y)}\pi(d \mid y_{1:k-1})\Bigg(\theta_k\rho_d - \sum_{x=k}^{K}\theta_x\rho_{y_x}\Bigg)}_{\text{expected direct reward minus the risk of placement}}\Bigg].\end{aligned} \tag{17}$$

In this formulation the gradient w.r.t. an item's log score $m(d)$ is weighted by the relevance after placement (excluding ρ_d) plus the expected direct reward ($\theta_k\rho_d$) minus the expected risk imposed by d before its placement. From Eq. 17 we can derive the novel *PL-Rank-2* estimator:

$$\begin{aligned}\frac{\delta}{\delta m}\mathcal{R}(q) \approx \frac{1}{N}\sum_{d\in\mathcal{D}}\Big[\frac{\delta}{\delta m}m(d)\Big]\sum_{i=1}^{N}\Bigg(\sum_{k=\text{rank}(d,y^{(i)})+1}^{K}\theta_k\rho_{y_k^{(i)}}\Bigg) \\ + \sum_{k=1}^{\text{rank}(d,y^{(i)})}\pi(d \mid y^{(i)}_{1:k-1})\Bigg(\theta_k\rho_d - \sum_{x=k}^{K}\theta_x\rho_{y_x^{(i)}}\Bigg).\end{aligned} \tag{18}$$

We will call this estimator: *PL-Rank-2*. Unlike PL-Rank-1 (Eq. 15), 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 $\mathcal{O}(N{\cdot}K{\cdot}D)$.

3.4 PL-Rank for Fairness

Importantly, PL-Rank can also estimate the gradient of a PL ranking model w.r.t. *exposure-based* fairness metrics [Diaz *et al.*, 2020; Biega *et al.*, 2018] Exposure represents the expected number of people that will examine an item, let θ_k be the probability that a user examines an item at rank k, then the exposure an item d receives under π is:

$$\mathcal{E}(q,d) = \mathbb{E}_y\Bigg[\sum_{k=1}^{K}\theta_k\mathbb{1}[y_k = d]\Bigg] = \sum_{y\in\pi}\pi(y)\sum_{k=1}^{K}\theta_k\mathbb{1}[y_k = d], \tag{19}$$

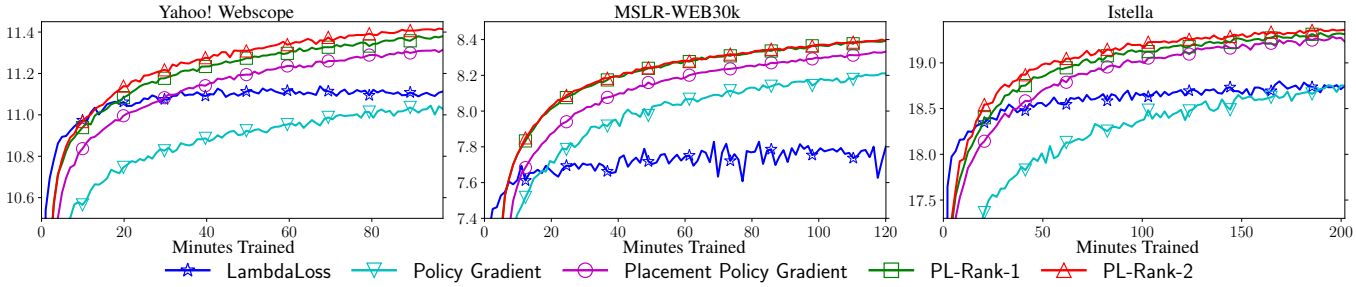

Figure 1: DCG@5 reached with different gradient estimation methods over training time, results are the mean of 20 independent runs.

where again for brevity we denote $\mathcal{E}_d = \mathcal{E}(q, d)$. Most fairness metrics for rankings consider whether exposure is distributed fairly over items, 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:

$$\frac{\delta}{\delta m}\mathcal{F}(q) = \sum_{d \in \mathcal{D}} \frac{\delta \mathcal{F}(q)}{\delta \mathcal{E}_d} \frac{\delta \mathcal{E}_d}{\delta m}. \tag{20}$$

To derive this gradient, we first note that the $\mathcal{E}_d$ (Eq. 19) 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. 17) accordingly, it will provide us the gradient $\frac{\delta \mathcal{E}_d}{\delta m}$. If we combine this fact with Eq. 20 we obtain the following PL-Rank-2 based gradient:

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

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. This potentially makes PL-Rank the first LTR method that is suited for optimizing both relevance and fairness.

4 Experimental Setup

Our experiments evaluate whether PL-Rank can reach higher performance in less computational time than policy gradients or LambdaLoss [Wang *et al.*, 2018]. We use the three largest publicly-available LTR industry datasets: *Yahoo! Webscope* [Chapelle and Chang, 2011], *MSLR-WEB30k* [Qin and Liu, 2013], and *Istella* [Dato *et al.*, 2016]. As a metric to optimize, we chose the commonly-used top-5 *Discounted Cumulative Gain* (DCG@5). To compare the computational costs of each method, we ran repeated experiments under identical circumstances on a single *Intel Xeon Silver 4214* CPU. We optimize neural networks with two hidden layers of 32 sigmoid activated nodes using standard stochastic gradient descent with a 0.01 learning rate. For calculating gradients we utilize *Tensorflow* [Abadi and others, 2016] with two exceptions: the sampling of rankings and $\frac{\delta \mathcal{R}(q)}{\delta m}$ with the PL-Rank algorithm are computed using *Numpy* [Harris and others, 2020]. The number of sampled ranked are scaled with epochs according to: $N = 10 + 90 \cdot \frac{\text{epoch}}{40}$. [1]

[1]Code: https://github.com/HarrieO/2021-SIGIR-plackett-luce.

5 Results

Figure 1 shows the performance of PL ranking models trained with PL-Rank, Policy Gradients and LambdaLoss over training time in minutes. We see that LambdaLoss, a method designed for deterministic ranking models, converges quickly but on suboptimal performance. The standard policy gradient (Eq. 7) appears to learn slowly, as we do not observe it to converge in our results. The placement policy gradient outperforms both baselines within 20 minutes of training across all three datasets. 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. To conclude, it thus appears that Pl-Rank-2 reaches higher performance faster when compared to the other methods in all tested cases, with the single exception of PL-Rank-1 on the MSLR dataset. Therefore, our results support the conclusion that PL-Rank-2 is the most time-efficient method when compared with policy gradients and LambdaLoss.

6 Conclusion

This paper addressed the optimization of PL-ranking models for both relevance and fairness ranking metrics. To alleviate the large computational costs of existing methods, 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. Our experimental results indicate that our novel methods considerably reduce the computational time required to reach optimal performance compared to existing 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.

References

[Abadi and others, 2016] Martín Abadi et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation OSDI'16)*, pages 265–283, 2016.

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

[Bruch *et al.*, 2019] Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and Marc Najork. 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*, pages 1241–1244, 2019.

[Bruch *et al.*, 2020] Sebastian Bruch, Shuguang Han, Michael Bendersky, and Marc Najork. A stochastic treatment of learning to rank scoring functions. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 61–69, 2020.

[Burges, 2010] Christopher J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, Microsoft, 2010.

[Cao *et al.*, 2007] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

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

[Dato *et al.*, 2016] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on Information Systems (TOIS)*, 35(2):Article 15, 2016.

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

[Fuhr, 1989] Norbert Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, 7(3):183–204, 1989.

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

[Harris and others, 2020] Charles R. Harris et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[Joachims, 2002] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

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

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

[Oosterhuis and de Rijke, 2018] Harrie Oosterhuis and Maarten de Rijke. Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1293–1302. ACM, 2018.

[Oosterhuis and de Rijke, 2020] Harrie Oosterhuis and Maarten de Rijke. 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, 2020.

[Paszke and others, 2019] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

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

[Qin and Liu, 2013] Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

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

[Ustimenko and Prokhorenkova, 2020] Aleksei Ustimenko and Liudmila Prokhorenkova. Stochasticrank: Global optimization of scale-free discrete functions. In *International Conference on Machine Learning*, pages 9669–9679. PMLR, 2020.

[Wang *et al.*, 2018] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1313–1322. ACM, 2018.

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

[Xia *et al.*, 2008] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199, 2008.