Differentiable Unbiased Online Learning to Rank

Harrie Oosterhuis
University of Amsterdam
oosterhuis@uva.nl

Maarten de Rijke
University of Amsterdam
derijke@uva.nl

ABSTRACT

Online Learning to Rank (OLTR) methods optimize rankers based on user interactions. State-of-the-art OLTR methods are built specifically for linear models. Their approaches do not extend well to non-linear models such as neural networks. We introduce an entirely novel approach to OLTR that constructs a weighted differentiable pairwise loss after each interaction: Pairwise Differentiable Gradient Descent (PDGD). PDGD breaks away from the traditional approach that relies on interleaving or multileaving and extensive sampling of models to estimate gradients. Instead, its gradient is based on inferring preferences between document pairs from user clicks and can optimize any differentiable model. We prove that the gradient of PDGD is unbiased w.r.t. user document pair preferences. Our experiments on the largest publicly available Learning to Rank (LTR) datasets show considerable and significant improvements under all levels of interaction noise. PDGD outperforms existing OLTR methods both in terms of learning speed as well as final convergence. Furthermore, unlike previous OLTR methods, PDGD also allows for non-linear models to be optimized effectively. Our results show that using a neural network leads to even better performance at convergence than a linear model. In summary, PDGD is an efficient and unbiased OLTR approach that provides a better user experience than previously possible.

CCS CONCEPTS

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

KEYWORDS

Learning to rank; Online learning; Gradient descent

ACM Reference Format:
Harrie Oosterhuis and Maarten de Rijke. 2018. Differentiable Unbiased Online Learning to Rank. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271686

1 INTRODUCTION

In order to benefit from unprecedented volumes of content, users rely on ranking systems to provide them with the content of their liking. Learning to Rank (LTR) in Information Retrieval (IR) concerns methods that optimize ranking models so that they order documents according to user preferences. In web search engines such models combine hundreds of signals to rank web-pages according to their relevance to user queries [21]. Similarly, ranking models are a vital part of recommender systems where there is no explicit search intent [18]. LTR is also prevalent in settings where other content is ranked, e.g., videos [5], products [19], conversations [28] or personal documents [38].

Traditionally, LTR has been applied in the *offline* setting where a dataset with annotated query-document pairs is available. Here, the model is optimized to rank documents according to the relevance annotations, which are based on the judgements of human annotators. Over time the limitations of this supervised approach have become apparent: annotated sets are expensive and time-consuming to create [4, 22]; when personal documents are involved such a dataset would breach privacy [38]; the relevance of documents to queries can change over time, like in a news search engine [8, 20]; and judgements of raters are not necessarily aligned with the actual users [31].

In order to overcome the issues with annotated datasets, previous work in LTR has looked into learning from user interactions. Work along these lines can be divided into *approaches that learn from historical interactions*, i.e., in the form of interaction logs [17], and *approaches that learn in an online setting* [39]. The latter regard methods that determine what to display to the user at each impression, and then immediately learn from observed user interactions and update their behavior accordingly. This online approach has the advantage that it does not require an existing ranker of decent quality, and thus can handle cold-start situations. Additionally, it is more responsive to the user by updating continuously and instantly, therefore allowing for a better experience. However, it is important that an online method can handle biases that come with user behavior: for instance, the observed interactions only take place with the displayed results, i.e., there is selection bias, and are more likely to occur with higher ranked items, i.e., there is position bias. Accordingly, a method should learn user preferences w.r.t. document relevance, and be robust to the forms of noise and bias present in the online setting. Overall, the online LTR approach promises to learn ranking models that are in line with user preferences, in a responsive matter, reaching good performance from few interactions, even in cold-start situations.

Despite these highly beneficial properties, previous work in Online Learning to Rank (OLTR) has only considered linear models [16, 33, 39] or trivial variants thereof [23]. The reason for this is that existing work in OLTR has worked with the Dueling Bandit Gradient Descent (DBGD) algorithm [39] as a basis. While very influential and effective, we identify two main problems with the gradient estimation of the DBGD algorithm:

(1) Gradient estimation is based on sampling model variants from a unit circle around the current model. This concept does not

extend well to non-linear models. Computing rankings for variants is also computationally costly for larger complex models.
(2) It uses online evaluation methods, i.e., interleaving or multileaving, to determine the gradient direction from the resulting set of models. However, these evaluation methods are designed for finding preferences between ranking systems, not (primarily) for determining how a model should be updated.

As an alternative we introduce *Pairwise Differentiable Gradient Descent* (PDGD), the first unbiased OLTR method that is applicable to any differentiable ranking model. PDGD infers pairwise document preferences from user interactions and constructs an unbiased gradient after each user impression. In addition, PDGD does not rely on sampling models for exploration, but instead models rankings as probability distributions over documents. Therefore, it allows the OLTR model to be very certain for specific queries and perform less exploration in those cases, while being much more explorative in other, uncertain cases. Our results show that, consequently, PDGD provides significant and considerable improvements over previous OLTR methods. This indicates that its gradient estimation is more in line with the preferences to be learned.

In this work, we answer the following three research questions:

RQ1 Does using PDGD result in significantly better performance than the current state-of-the-art Multileave Gradient Descent?
RQ2 Is the gradient estimation of PDGD unbiased?
RQ3 Is PDGD capable of effectively optimizing different types of ranking models?

To facilitate replicability and repeatability of our findings, we provide open source implementations of PDGD and our experiments under the permissive MIT open-source license.[1]

2 RELATED WORK

2.1 Learning to rank

LTR can be applied to the offline and online setting. In the offline setting LTR is approached as a supervised problem where the relevance of each query-document pair is known. Most of the challenges with offline LTR come from obtaining annotations. For instance, gathering annotations is time-consuming and expensive [4, 22, 27]. Furthermore, in privacy sensitive-contexts it would be unethical to annotate items, e.g., for personal emails or documents [38]. Moreover, for personalization problems annotators are unable to judge what specific users would prefer. Also, (perceived) relevance chances over time, due to cognitive changes on the user's end [37] or due to changes in document collections [8] or the real world [20]. Finally, annotations are not necessarily aligned with user satisfaction, as judges may interpret queries differently from actual users [31]. Consequently, the limitations of offline LTR have led to an increased interest in alternative approaches to LTR.

2.2 Online learning to rank

OLTR is an attractive alternative as it learns directly from interacting with users [39]. By doing so it attempts to solve the issues with offline annotations that occur in LTR, as user preferences are expected to be better represented by interactions than with offline annotations [30]. Unlike methods in the offline setting, OLTR algorithms have to simultaneously perform ranking while also optimizing their ranking model. In other words, an OLTR algorithm decides what rankings to display to users, while at the same time learning from the interactions with the presented rankings. While the potential of learning in the online setting is great, it has its own challenges. In particular, the main difficulties of the OLTR task are *bias* and *noise*. Any user interaction that does not reflect their true preference is considered noise, this happens frequently e.g., clicks often occur for unexpected reasons [31]. Bias comes in many forms, for instance, selection bias occurs because interactions only involve displayed documents [38]. Another common bias is position bias, a consequence from the fact documents at the top of a ranking are more likely to be considered [40]. An OLTR method should thus take into account the biases that affect user behavior while also being robust to noise, in order to learn the *true* user preferences.

OLTR methods can be divided into two groups [41]: *model-based* methods that learn the best ranked list under some model of user interaction with the list [29, 35], such as a click model [6], and *model-free* algorithms that learn the best ranker in a family of rankers [13, 39]. Model-based methods may have greater statistical efficiency but they give up generality, essentially requiring us to learn a separate model for every query. For the remainder of this paper, we focus on model-free OLTR methods.

2.3 DBGD and beyond

State-of-the-art (model-free) OLTR approaches learn user preferences by approaching optimization as a dueling bandit problem [39]. They estimate the gradient of the model w.r.t. user satisfaction by comparing the current model to sampled variations of the model. The original DBGD algorithm [39] uses interleaving methods to make these comparisons: at each interaction the rankings of two rankers are combined to create a single result list. From a large number of clicks on such a combined result list a user preference between the two rankers can reliably be inferred [15]. Conversely, DBGD compares its current ranking model to a different slight variation at each impression. Then, if a click is indicative of a preference for the variation, the current model is slightly updated towards it. Accordingly, the model of DBGD will continuously update itself and oscillate towards an inferred optimum.

Other work in OLTR has used DBGD as a basis and extended upon it. Notably, Hofmann et al. [13] have proposed a method that guides exploration by only sampling variations that seem promising from historical interaction data. Unfortunately, while this approach provides faster initial learning, the historical data introduces bias which leads to the quality of the ranking model to steadily decrease over time [25]. Alternatively, Schuth et al. [33] introduced Multileave Gradient Descent (MGD), this extension replaced the interleaving of DBGD with multileaving methods. In turn the multileaving paradigm is an extension of interleaving where a set of rankers are compared efficiently [24, 32, 34]. Conversely, multileaving methods can combine the rankings of more than two rankers and thus infer preferences over a set of rankers from a single click. MGD uses this property to estimate the gradient more effectively

[1]https://github.com/HarrieO/OnlineLearningToRank

by comparing a large number of model variations per user impression [25, 33]. As a result, MGD requires fewer user interactions to converge on the same level of performance as DBGD. Another alternative approach was considered by Hofmann et al. [14], who inject the ranking from the current model with randomly sampled documents. Then, after each user impression, a pairwise loss is constructed from inferred preferences between documents. This pairwise approach was not found to be more effective than DBGD.

Quite remarkably, all existing work in OLTR has only considered linear models. Recently, Oosterhuis and de Rijke [23] recognized that a tradeoff unique to OLTR arises when choosing models. High capacity models such as neural networks [2] require more data than simpler models. On the one hand, this means that high capacity models need more user interactions to reach the same level of performance, thus giving a worse initial user experience. On the other hand, high capacity models are capable of finding better optima, thus lead to better final convergence and a better long-term user experience. This dilemma is named the *speed-quality* tradeoff, and as a solution a cascade of models can be optimized: combining the initial learning speed of a simple model with the convergence of a complex one. But there are more reasons why non-linear models have so far been absent from OLTR. Importantly, the DBGD was designed for linear models from the ground up; relying on a unit circle to sample model variants and averaging models to estimate the gradient. Furthermore, the computational cost of maintaining an extensive set of model variants for large and complex models makes this approach very impractical.

Our contribution over the work listed above is an OLTR method that is not an extension of DBGD, instead it computes differentiable pairwise loss to update its model. Unlike the existing pairwise approach, our loss function is unbiased and our exploration is performed using the model's confidence over documents. Finally, we also show that this is the first OLTR method to effectively optimize neural networks in the online setting.

3 METHOD

In this section we introduce a novel OLTR algorithm: PDGD. First, Section 3.1 describes PDGD in detail, before Section 3.2 formalizes and proves the unbiasedness of the method. Table 1 lists the notation we use.

3.1 Pairwise Differentiable Gradient Descent

PDGD revolves around optimizing a ranking model $f_\theta(\mathbf{d})$ that takes a feature representation of a query-document pair $\mathbf{d}$ as input and outputs a score. The aim of the algorithm is to find the parameters θ so that sorting the documents by their scores in descending order provides the most optimal rankings. Because this is an online algorithm, the method must first decide what ranking to display to the user, then after the user has interacted with the displayed ranking, it may update θ accordingly.

Unlike previous OLTR approaches, PDGD does not rely on any online evaluation methods. Instead, a Plackett-Luce (PL) model is applied to the ranking function $f_\theta(\cdot)$ resulting in a distribution over the document set D:

$$P(d|D) = \frac{e^{f_\theta(\mathbf{d})}}{\sum_{d' \in D} e^{f_\theta(\mathbf{d}')}}. \tag{1}$$

Table 1: Main notation used in the paper.

Notation	Description
d, d_k, d_l	document
$\mathbf{d}$	feature representation of a query-document pair
D	set of documents
R	ranked list
R^*	the reversed pair ranking $R^*(d_k, d_l, R)$
R_i	document placed at rank i
ρ	preference pair weighting function
θ	parameters of the ranking model
$f_\theta(\cdot)$	ranking model with parameters θ
$f(\mathbf{d}_k)$	ranking score for a document from model
$click(d)$	a click on document d
$d_k =_{rel} d_l$	two documents equally preferred by users
$d_k >_{rel} d_l$	a user preference between two documents
$d_k >_{\mathbf{c}} d_l$	document preference inferred from clicks

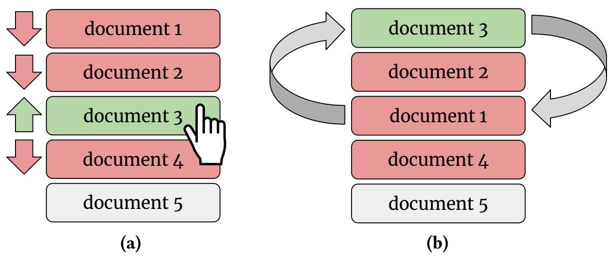

Figure 1: Left: a click on a document ranking R and the inferred preferences of d_3 over $\{d_1, d_2, d_4\}$. Right: the reversed pair ranking $R^*(d_1, d_3, R)$ for the document pair d_1 and d_3.

A ranking R to display to the user is then created by sampling from the distribution k times, where after each placement the distribution is renormalized to prevent duplicate placements. PL models have been used before in LTR. For instance, the ListNet method [3] optimizes such a model in the offline setting. With R_i denoting the document at position i, the probability of the ranking R then becomes:

$$P(R|D) = \prod_{i=1}^{k} P(R_i | D \setminus \{R_1, \ldots, R_{i-1}\}). \tag{2}$$

After the ranking R has been displayed to the user, they have the option to interact with it. The user may choose to click on some (or none) of the documents. Based on these clicks, PDGD will infer preferences between the displayed documents. We assume that clicked documents are preferred over observed unclicked documents. However, to the algorithm it is unknown which unclicked documents the user has considered. As a solution, PDGD relies on the assumption that every document preceding a clicked document and the first subsequent unclicked document was observed, as illustrated in Figure 1a. This preference assumption has been proven useful in IR before, for instance in pairwise LTR on click logs [17] and recently in online evaluation [24]. We will denote preferences between documents inferred from clicks as: $d_k >_{\mathbf{c}} d_l$ where d_k is preferred over d_l.

Algorithm 1 Pairwise Differentiable Gradient Descent (PDGD).

```
1:  **Input**: initial weights: $\theta_1$; scoring function: $f$; learning rate $\eta$.
2:  **for** $t \leftarrow 1 \ldots \infty$ **do**
3:      $q_t \leftarrow receive\_query(t)$                         // obtain a query from a user
4:      $D_t \leftarrow preselect\_documents(q_t)$                 // preselect documents for query
5:      $\mathbf{R}_t \leftarrow sample\_list(f_{\theta_t}, D_t)$  // sample list according to Eq. 1
6:      $\mathbf{c}_t \leftarrow receive\_clicks(\mathbf{R}_t)$    // show result list to the user
7:      $\nabla f_{\theta_t} \leftarrow \mathbf{0}$                // initialize gradient
8:      **for** $d_k >_{\mathbf{c}} d_l \in \mathbf{c}_t$ **do**
9:          $w \leftarrow \rho(d_k, d_l, R, D)$                     // initialize pair weight (Eq. 5)
10:         $w \leftarrow w \frac{e^{f_{\theta_t}(\mathbf{d}_k)} e^{f_{\theta_t}(\mathbf{d}_l)}}{(e^{f_{\theta_t}(\mathbf{d}_k)} + e^{f_{\theta_t}(\mathbf{d}_l)})^2}$   // pair gradient (Eq. 4)
11:         $\nabla f_{\theta_t} \leftarrow \nabla \theta_t + w(f'_{\theta_t}(\mathbf{d}_k) - f'_{\theta_t}(\mathbf{d}_l))$   // model gradient (Eq. 4)
12:     $\theta_{t+1} \leftarrow \theta_t + \eta \nabla f_{\theta_t}$   // update the ranking model
```

Then θ is updated by optimizing pairwise probabilities over the preference pairs; for each inferred document preference $d_k >_{\mathbf{c}} d_l$, the probability that the preferred document d_k is sampled before d_l is sampled is increased [36]:

$$P(d_k \succ d_k) = \frac{P(d_k|D)}{P(d_k|D) + P(d_l|D)} = \frac{e^{f(\mathbf{d}_k)}}{e^{f(\mathbf{d}_k)} + e^{f(\mathbf{d}_l)}}. \tag{3}$$

We have chosen for pairwise optimization over listwise optimization because a pairwise method can be made unbiased by reweighing preference pairs. To do this we introduce the weighting function $\rho(d_k, d_l, R, D)$ and estimate the gradient of the user preferences by the weighted sum:

$$\begin{aligned} &\nabla f_\theta(\cdot) \\ &\approx \sum_{d_k >_{\mathbf{c}} d_l} \rho(d_k, d_l, R, D) \left[\nabla P(d_k \succ d_l)\right] \\ &= \sum_{d_k >_{\mathbf{c}} d_l} \rho(d_k, d_l, R, D) \frac{e^{f_\theta(\mathbf{d}_k)} e^{f_\theta(\mathbf{d}_l)}}{(e^{f_\theta(\mathbf{d}_k)} + e^{f_\theta(\mathbf{d}_l)})^2} \left(f'_\theta(\mathbf{d}_k) - f'_\theta(\mathbf{d}_l)\right). \end{aligned} \tag{4}$$

The ρ function is based on the reversed pair ranking $R^*(d_k, d_l, R)$, which is the same ranking as R with the position of d_k and d_l swapped. An example of a reversed pair ranking is illustrated in Figure 1b. The idea is that if a preference for $d_k >_{\mathbf{c}} d_l$ is inferred in R and both documents are equally relevant, then the reverse preference $d_l >_{\mathbf{c}} d_k$ is equally likely to be inferred in $R^*(d_k, d_l, R)$. The ρ function reweighs the found preferences to the ratio between the probabilities of R or $R^*(d_k, d_l, R)$ occurring:

$$\rho(d_k, d_l, R, D) = \frac{P(R^*(d_k, d_l, R)|D)}{P(R|D) + P(R^*(d_k, d_l, R)|D)}. \tag{5}$$

This procedure has similarities with importance sampling [26]; however, we found that reweighing according to the ratio between R and R^* provides a more stable performance, since it produces less extreme values. Section 3.2 details exactly how ρ creates an unbiased gradient.

Algorithm 1 describes the PDGD method step by step: Given the initial parameters θ_1 and a differentiable scoring function f (Line 1), the method waits for a user-issued query q_t to arrive (Line 3). Then the preselected set of documents D_t for the query is fetched (Line 4), in our experiments these preselections are given in the LTR datasets that we use. A result list R is sampled from the current model (Line 5 and Equation 1) and displayed to the user. The clicks from the user are logged (Line 6) and preferences between the displayed documents inferred (Line 8). The gradient is initialized (Line 7), and for each pair document pair d_k, d_l such that $d_k >_{\mathbf{c}} d_l$, the weight $\rho(d_k, d_l, R, D)$ is calculated (Line 9 and Equation 5), followed by the gradient for the pair probability (Line 10 and Equation 4). Finally, the gradient for the scoring function f is weighted and added to the gradient (Line 11), resulting in the estimated gradient. The model is then updated by taking an η step in the direction of the gradient (Line 12). The algorithm again waits for the next query to arrive and thus the process continues indefinitely.

PDGD has some notable advantages over Multileave Gradient Descent (MGD) [33]. Firstly, it explicitly models uncertainty over the documents per query, thus PDGD is able to have high confidence in its ranking for one query, while being completely uncertain for another query. As a result, it will vary the amount of exploration per query, allowing it to avoid exploration in cases where it is not required and focussing on areas where it can improve. In contrast, MGD does not explicitly model confidence: its degree of exploration is only affected by the norm of its linear model [23]. Consequently, MGD is unable to vary exploration per query nor is there a way to directly measure its level of confidence. Secondly, PDGD works for any differentiable scoring function f and does not rely on sampling model variants. Conversely, MGD is based around sampling from the unit sphere around a model; this approach is very ineffective for non-linear models. Additionally, sampling large models and producing rankings for them can be very computationally expensive. Besides these beneficial properties, our experimental results in Section 5 show that PDGD achieves significantly higher levels of performance than MGD and other previous methods.

3.2 Unbiased gradient estimation

The previous section introduced PDGD; this section answers **RQ2**: is the gradient estimation of PDGD unbiased?

First, Theorem 3.1 will provide a definition of unbiasedness w.r.t. user document pair preferences. Then we state the assumptions we make about user behavior and use them to prove Theorem 3.1.

Theorem 3.1. *The expected estimated gradient of PDGD can be written as a weighted sum, with a unique weight $\alpha_{k,l}$ for each possible document pair d_k and d_l in the document collection D:*

$$E[\nabla f_\theta(\cdot)] = \sum_{d_k, d_l \in D} \alpha_{k,l}(f'_{\theta_t}(\mathbf{d}_k) - f'_{\theta_t}(\mathbf{d}_l)). \tag{6}$$

The signs of the weights $\alpha_{k,l}$ adhere to user preferences between documents. That is, if there is no preference:

$$d_k =_{rel} d_l \Leftrightarrow \alpha_{k,l} = 0; \tag{7}$$

if d_k is preferred over d_l:

$$d_k >_{rel} d_l \Leftrightarrow \alpha_{k,l} > 0; \tag{8}$$

and if d_l is preferred over d_k:

$$d_k <_{rel} d_l \Leftrightarrow \alpha_{k,l} < 0. \tag{9}$$

Therefore, in expectation PDGD will perform updates that adhere to the preferences between the documents in every possible document pair.

Assumptions. To prove Theorem 3.1 the following assumptions about user behavior will be used:

Assumption 1. We assume that clicks from a user are position biased and conditioned on the relevance of the current document and the previously considered documents. For a click on a document in ranking R at position i the probability can be written as:

$$P(click(R_i)|\{R_0, \dots, R_{i-1}, R_{i+1}\}). \tag{10}$$

For ease of notation, we will denote the set of "other documents" as $\{\dots\}$ from here on.

Assumption 2. If there is no user preference between two documents d_k, d_l, denoted by $d_k =_{rel} d_l$, we assume that each is equally likely to be clicked given the same context:

$$d_k =_{rel} d_l \Rightarrow P(click(d_k)|\{\dots\}) = P(click(d_l)|\{\dots\}). \tag{11}$$

Assumption 3. If a document in the set of documents being considered is replaced with an equally preferred document the click probability is not affected:

$$d_k =_{rel} d_l \Rightarrow P(click(R_i)|\{\dots, d_k\}) = P(click(R_i)|\{\dots, d_l\}). \tag{12}$$

Assumption 4. Similarly, given the same context if one document is preferred over another, then it is more likely to be clicked:

$$d_k >_{rel} d_l \Rightarrow P(click(d_k)|\{\dots\}) > P(click(d_l)|\{\dots\}). \tag{13}$$

Assumption 5. Lastly, for any pair $d_k >_{rel} d_l$, the considered document set $\{\dots, d_k\}$ and the same set with d_k replaced $\{\dots, d_l\}$. We assume that the preferred d_k in the context of $\{\dots, d_l\}$ is more likely to be clicked than d_l in the context of $\{\dots, d_k\}$:

$$d_k >_{rel} d_l \Rightarrow P(click(d_k)|\{\dots, d_k\}) > P(click(d_l)|\{\dots, d_l\}). \tag{14}$$

These are all the assumptions we make about the user. With these assumptions, we can proceed to prove Theorem 3.1.

Proof of Theorem 3.1. We denote the probability of inferring the preference of d_k over d_l in ranking R as $P(d_k >_{\mathbf{c}} d_l|R)$. Then the expected gradient $\nabla f_\theta(\cdot)$ of PDGD can be written as:

$$E[\nabla f_\theta(\cdot)] = \sum_R \sum_{d_k, d_l \in D} \Big[P(d_k >_{\mathbf{c}} d_l|R) \cdot P(R) \cdot \rho(d_k, d_l, R, D) \cdot \ [\nabla P(d_k \succ d_l)] \Big]. \tag{15}$$

We will rewrite this expectation using the symmetry property of the reversed pair ranking:

$$R^n = R^*(d_k, d_l, R^m) \Leftrightarrow R^m = R^*(d_k, d_l, R^n). \tag{16}$$

First, we define a weight $\omega^R_{k,l}$ for every document pair d_k, d_l and ranking R so that:

$$\begin{aligned}\omega^R_{k,l} &= P(R)\rho(d_k, d_l, R, D) \\ &= \frac{P(R|D)P(R^*(d_k, d_l, R)|D)}{P(R|D) + P(R^*(d_k, d_l, R)|D)}.\end{aligned} \tag{17}$$

Therefore, the weight for the reversed pair ranking is equal:

$$\begin{aligned}\omega^{R^*(d_k,d_l,R)}_{k,l} &= P(R^*(d_k, d_l, R))\rho(d_k, d_l, R^*(d_k, d_l, R), D) \\ &= \omega^R_{k,l}.\end{aligned} \tag{18}$$

Then, using the symmetry of Equation 3 we see that:

$$\nabla P(d_k \succ d_l) = -\nabla P(d_l \succ d_k). \tag{19}$$

Thus, with R^* as a shorthand for $R^*(d_k, d_l, R)$, the expectation can be rewritten as:

$$E[\nabla f_\theta(\cdot)] = \sum_{d_k, d_l \in D} \sum_R \frac{\omega^R_{i,j}}{2} \left(P(d_k >_{\mathbf{c}} d_l|R) - P(d_l >_{\mathbf{c}} d_k|R^*) \right) \cdot \Big[\nabla P(d_k \succ d_l) \Big], \tag{20}$$

proving that the expected gradient matches the form of Equation 6. Then to prove that Equations 7, 8, and 9 are correct we will show that:

$$d_k =_{rel} d_l \Rightarrow P(d_k >_{\mathbf{c}} d_l|R) = P(d_l >_{\mathbf{c}} d_k|R^*), \tag{21}$$
$$d_k >_{rel} d_l \Rightarrow P(d_k >_{\mathbf{c}} d_l|R) > P(d_l >_{\mathbf{c}} d_k|R^*), \tag{22}$$
$$d_k <_{rel} d_l \Rightarrow P(d_k >_{\mathbf{c}} d_l|R) < P(d_l >_{\mathbf{c}} d_k|R^*). \tag{23}$$

If a preference $R_i >_{\mathbf{c}} R_j$ is inferred then there are only three possible cases based on the positions:

(1) The clicked document succeeds the unclicked document by more than one position: $i + 1 > j$.
(2) The clicked document precedes the unclicked document by more than one position: $i - 1 < j$.
(3) The clicked document is one position before or after the unclicked document: $i = j + 1 \lor i = j - 1$.

In the first case the clicked document succeeds the other by more than one position, the probability of an inferred preference is then:

$$\begin{aligned} i + 1 > j \Rightarrow P(R_i >_{\mathbf{c}} R_j|R) = \\ P(\mathbf{c}_i|R_i, \{\dots, R_j\}\ ) \cdot \\ (1 - P(\mathbf{c}_j|R_j, \{\dots\})). \end{aligned} \tag{24}$$

Combining Assumption 2 and 3 with Equation 24 proves Equation 21 for this case. Furthermore, combining Assumption 4 and 5 with Equation 24 proves Equations 22 and 23 for this case as well.

Then the second case is when the clicked document appears more than one position before the unclicked document, the probability of the inferred preference is then:

$$\begin{aligned} i + 1 < j \Rightarrow P(R_i >_{\mathbf{c}} R_j|R) = \\ P(\mathbf{c}_i|R_i, \{\dots\}) \cdot \\ (1 - P(\mathbf{c}_j|R_j, \{\dots, R_i\}\ )) \cdot \\ P(\mathbf{c}_{rem}), \end{aligned} \tag{25}$$

where $P(\mathbf{c}_{rem})$ denotes the probability of an additional click that is required to add R_j to the inferred observed documents. First, due to Assumption 1 this probability will be the same for R and R^*:

$$P(\mathbf{c}_{rem}|R_i, R_j, R) = P(\mathbf{c}_{rem}|R_i, R_j, R^*). \tag{26}$$

Combining Assumption 2 and 3 with Equation 25 also proves Equation 21 for this case. Furthermore, combining Assumption 4 and 5 with Equation 25 also proves Equation 22 and 23 for this case as well.

Table 2: Instantiations of Cascading Click Models [10] as used for simulating user behavior in experiments.

	$P(click = 1 \mid R)$					$P(stop = 1 \mid click = 1, R)$				
R	0	1	2	3	4	0	1	2	3	4
perf	0.0	0.2	0.4	0.8	1.0	0.0	0.0	0.0	0.0	0.0
nav	0.05	0.3	0.5	0.7	0.95	0.2	0.3	0.5	0.7	0.9
inf	0.4	0.6	0.7	0.8	0.9	0.1	0.2	0.3	0.4	0.5

Lastly, in the third case the clicked document is one position before or after the other document, the probability of the inferred preference is then:

$$\begin{aligned} i = j + 1 \lor i = j - 1 \Rightarrow P(R_i >_{\mathbf{c}} R_j | R) = \\ P(\mathbf{c}_i | R_i, \{\ldots, R_j\}\ ) \cdot \\ (1 - P(\mathbf{c}_j | R_j, \{\ldots, R_i\})). \end{aligned} \tag{27}$$

Combining Assumption 3 with Equation 27 proves Equation 21 for this case as well. Then, combining Assumption 5 with Equation 27 also proves Equation 22 and 23 for this case. □

This concludes our proof of the unbiasedness of PDGD. Hence, we answer **RQ2** positively: the gradient estimation of PDGD is unbiased. We have shown that the expected gradient is in line with user preferences between document pairs.

4 EXPERIMENTS

In this section we detail the experiments that were performed to answer the research questions in Section 1.

4.1 Datasets

Our experiments are performed over five publicly available LTR datasets; we have selected three large labelling sets from commercial search engines and two smaller research datasets. Every dataset consists of a set of queries with each query having a corresponding preselected document set. The exact content of the queries and documents are unknown, each query is represented only by an identifier, but each query-document pair has a feature representation and relevance label. Depending on the dataset, the relevance labels are graded differently; we have purposefully chosen datasets that have at least two grades of relevance. Each dataset is divided in training, validation and test partitions.

The oldest datasets we use are *MQ2007* and *MQ2008* [27] which are based on the Million Query Track [1] and consist of 1,700 and 800 queries. They use representations of 46 features that encode ranking models such as TF.IDF, BM25, Language Modeling, PageRank, and HITS on different parts of the documents. They are divided into five folds and the labels are on a three-grade scale from *not relevant* (0) to *very relevant* (2).

In 2010 Microsoft released the *MSLR-WEB30k* and *MLSR-WEB10K* datasets [27], which are both created from a retired labelling set of a commercial web search engine (Bing). The former contains 30,000 queries with each query having 125 assessed documents on average, query-document pairs are encoded in 136 features, The latter is a subsampling of 10,000 queries from the former dataset. For practical reasons only *MLSR-WEB10K* was used for this paper. Also in 2010 Yahoo! released an LTR dataset [4]. It consists of 29,921 queries and 709,877 documents encoded in 700 features, all sampled from query logs of the Yahoo! search engine. Finally, in 2016 a LTR dataset released by the Istella search engine [7]. It is the largest with 33,118 queries, an average of 315 documents per query and 220 features. These three commercial datasets all label relevance on a five-grade scale: from *not relevant* (0) to *perfectly relevant* (4).

4.2 Simulating user behavior

For simulating users we follow the standard setup for OLTR simulations [11, 14, 25, 33, 42]. First, queries issued by users are simulated by uniformly sampling from the static dataset. Then the algorithm determines the result list of documents to display. User interactions with the displayed list are then simulated using a *cascade click model* [6, 10]. This models a user who goes through the documents one at a time in the displayed order. At each document, the user decides whether to click it or not, modelled as a probability conditioned on the relevance label R: $P(click = 1 \mid R)$. After a click has occurred, the user's information need may be satisfied and they may then stop considering documents. The probability of a user stopping after a click is modelled as $P(stop = 1 \mid click = 1, R)$. For our experiments $\kappa = 10$ documents are displayed at each impression.

The three instantiations of cascade click models that we used are listed in Table 2. First, a *perfect* user is modelled who considers every document and solely clicks on all relevant documents. The second models a user with a *navigational* task, where a single highly relevant document is searched. Finally, an *informational* instantiation models a user without a specific information need, and thus typically clicks on many documents. These models have varying levels of noise, as each behavior depends on the relevance labels of documents with a different degree.

4.3 Experimental runs

For our experiments three baselines are used. First, MGD with Probabilistic Multileaving [25]; this is the highest performing existing OLTR method [23, 25]. For this work $n = 49$ candidates were sampled per iteration from the unit sphere with $\delta = 1$; updates are performed with $\eta = 0.01$ and zero initialization was used. Additionally, DBGD is used for comparison as it is one of the most influential methods, it was run with the same parameters except that only $n = 1$ candidate is sampled per iteration. Furthermore, we also let DBGD optimize a single hidden-layer neural network with 64 hidden nodes and sigmoid activation functions with *Xavier* initialization [9]. These parameters were also found most effective in previous work [14, 25, 33, 39].

Additionally, the pairwise method introduced by Hofmann et al. [14] is used as a baseline. Despite not showing significant improvements over DBGD in the past [14], the comparison with PDGD is interesting because they both estimate gradients from pairwise preferences. For this baseline, $\eta = 0.01$ and $\epsilon = 0.8$ is used; these parameters are chosen to maximize the performance at convergence [14].

Runs with PDGD are performed with both a linear and neural ranking model. For the linear ranking model $\eta = 0.1$ and zero initialization was used. The neural network has the same parameters as the one optimized by DBGD, except for $\eta = 0.1$.

Table 3: Offline performance (NDCG) for different instantiations of CCM (Table 2). The standard deviation is shown in brackets, bold values indicate the highest performance per dataset and click model, significant improvements over the DBGD, MGD and pairwise baselines are indicated by △ ($p < 0.05$) and ▲ ($p < 0.01$), no losses were measured.

	MQ2007	MQ2008	MSLR-WEB10k	Yahoo	istella
			perfect		
DBGD (linear)	0.483 (0.023)	0.683 (0.024)	0.331 (0.010)	0.684 (0.010)	0.448 (0.014)
DBGD (neural)	0.463 (0.025)	0.670 (0.026)	0.319 (0.014)	0.676 (0.016)	0.429 (0.017)
MGD (linear)	0.494 (0.022)	0.690 (0.019)	0.333 (0.003)	0.714 (0.002)	0.496 (0.004)
Pairwise (linear)	0.479 (0.022)	0.674 (0.017)	0.315 (0.003)	0.709 (0.001)	0.252 (0.002)
PDGD (linear)	**0.511 (0.017)** ▲ ▲ ▲ ▲	**0.699 (0.024)** ▲ ▲ ▲ ▲	0.427 (0.005) ▲ ▲ ▲ ▲	**0.736 (0.004)** ▲ ▲ ▲ ▲	0.573 (0.004) ▲ ▲ ▲ ▲
PDGD (neural)	0.509 (0.020) ▲ ▲ ▲ ▲	0.698 (0.024) ▲ ▲ ▲ ▲	**0.430 (0.006)** ▲ ▲ ▲ ▲	0.733 (0.005) ▲ ▲ ▲ ▲	**0.575 (0.006)** ▲ ▲ ▲ ▲
			navigational		
DBGD (linear)	0.461 (0.025)	0.670 (0.025)	0.319 (0.011)	0.661 (0.023)	0.401 (0.015)
DBGD (neural)	0.430 (0.033)	0.646 (0.031)	0.304 (0.019)	0.649 (0.029)	0.382 (0.024)
MGD (linear)	0.426 (0.020)	0.662 (0.015)	0.321 (0.003)	0.706 (0.009)	0.405 (0.004)
Pairwise (linear)	0.476 (0.022)	0.677 (0.018)	0.312 (0.003)	0.696 (0.004)	0.209 (0.002)
PDGD (linear)	**0.496 (0.019)** ▲ ▲ ▲ ▲	**0.695 (0.021)** ▲ ▲ ▲ ▲	**0.406 (0.015)** ▲ ▲ ▲ ▲	**0.725 (0.005)** ▲ ▲ ▲ ▲	**0.540 (0.008)** ▲ ▲ ▲ ▲
PDGD (neural)	0.493 (0.020) ▲ ▲ ▲ ▲	0.692 (0.019) ▲ ▲ ▲ ▲	0.386 (0.019) ▲ ▲ ▲ ▲	0.722 (0.006) ▲ ▲ ▲ ▲	0.532 (0.011) ▲ ▲ ▲ ▲
			informational		
DBGD (linear)	0.411 (0.036)	0.631 (0.036)	0.299 (0.017)	0.620 (0.035)	0.360 (0.028)
DBGD (neural)	0.383 (0.047)	0.595 (0.053)	0.276 (0.033)	0.603 (0.040)	0.316 (0.057)
MGD (linear)	0.406 (0.021)	0.647 (0.036)	0.318 (0.003)	0.676 (0.043)	0.387 (0.005)
Pairwise (linear)	0.478 (0.022)	0.677 (0.018)	0.311 (0.003)	0.690 (0.006)	0.183 (0.001)
PDGD (linear)	**0.487 (0.021)** ▲ ▲ ▲ ▲	**0.690 (0.022)** ▲ ▲ ▲ ▲	**0.368 (0.025)** ▲ ▲ ▲ ▲	**0.713 (0.008)** ▲ ▲ ▲ ▲	**0.532 (0.010)** ▲ ▲ ▲ ▲
PDGD (neural)	0.483 (0.022) ▲ ▲ ▲	0.686 (0.022) ▲ ▲ ▲ ▲	0.355 (0.021) ▲ ▲ ▲ ▲	0.709 (0.009) ▲ ▲ ▲ ▲	0.525 (0.012) ▲ ▲ ▲ ▲

4.4 Metrics and tests

Two aspects of performance are evaluated seperately: the final convergence and the ranking quality during training.

Final convergence is addressed in *offline performance* which is the average NDCG@10 of the ranking model over the queries in the held-out test-set. The offline performance is measured after 10,000 impressions at which point most ranking models have reached convergence. The user experience during optimization should be considered as well, since deterring users during training would compromise the goal of OLTR. To address this aspect of evaluation *online performance* has been introduced [12]; it is the cumulative discounted NDCG@10 of the rankings displayed during training. For T sequential queries with R^t as the ranking displayed to the user at timestep t, this is:

$$Online\_Performance = \sum_{t=1}^{T} NDCG(R^t) \cdot \gamma^{(t-1)}. \tag{28}$$

This metric models the expected reward a user receives with a γ probability that the user stops searching after each query. We follow previous work [23, 25] by choosing a discount factor of $\gamma = 0.9995$, consequently queries beyond the horizon of 10,000 queries have a less than 1% impact.

Lastly, all experimental runs are repeated 125 times, spread evenly over the available dataset folds. Results are averaged and a two-tailed Student's t-test is used for significance testing. In total, our results are based on more than 90,000,000 user impressions.

5 RESULTS AND ANALYSIS

Our main results are displayed in Table 3 and Table 4, showing the offline and online performance of all methods, respectively. Additionally, Figure 2 displays the offline performance on the MSLR-WEB10k dataset over 30,000 impressions and Figure 3 over 1,000,000 impressions. We use these results to answer **RQ1** – whether PDGD provides significant improvements over existing OLTR methods – and **RQ3** – whether PDGD is successful at optimizing different types of ranking models.

5.1 Convergence of ranking models

First, we consider the offline performance after 10,000 impressions as reported in Table 3. We see that the DBGD and MGD baselines reach similar levels of performance, with marginal differences at low levels of noise. Our results seem to suggest that MGD provides an efficient alternative to DBGD that requires fewer user interactions and is more robust to noise. However, MGD does not appear to have an improved point of convergence over DBGD, Figure 2 further confirms this conclusion. Additionally, Table 3 and Figure 3 reveal thats DBGD is incapable of training its neural network so that it improves over the linear model, even after 1,000,000 impressions.

Alternatively, the pairwise baseline displays different behavior, providing improvements over DBGD and MGD on most datasets under all levels of noise. However, on the istella dataset large decreases in performance are observed. Thus it is unclear if this method provides a reliable alternative to DBGD or MGD in terms of convergence. Figure 2 also reveals that it converges within several

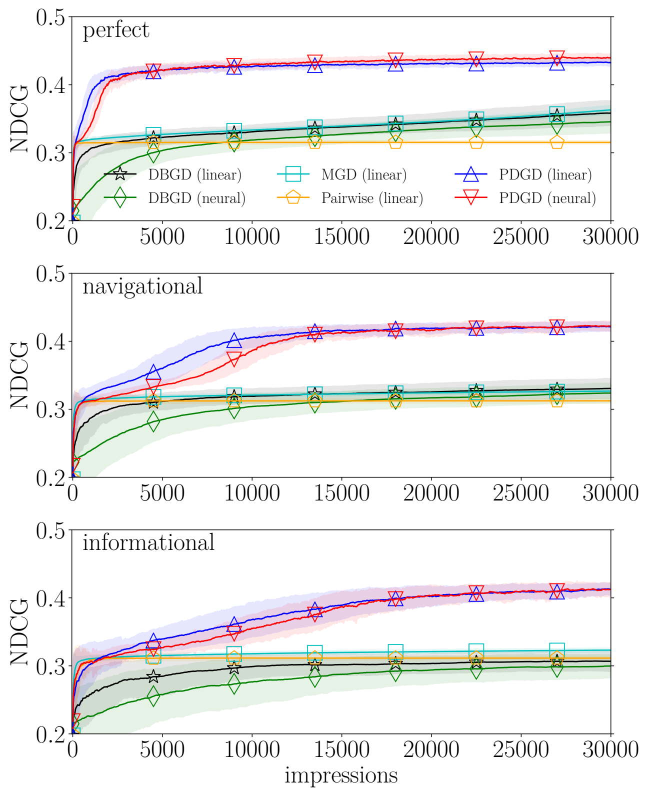

Figure 2: Offline performance (NDCG) on the MSLR-WEB10k dataset under three different click models, the shaded areas indicate the standard deviation.

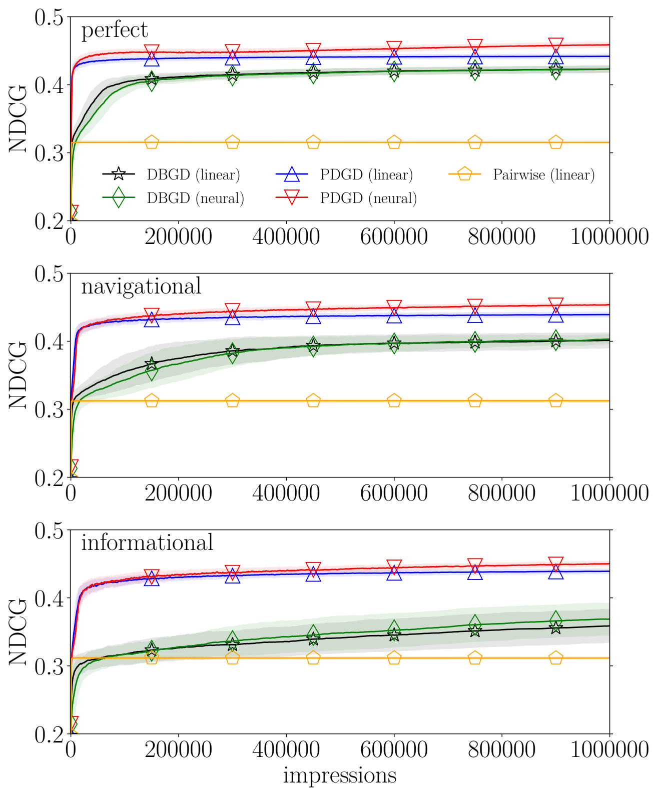

Figure 3: Long-term offline performance (NDCG) on the MSLR-WEB10k dataset under three click models, the shaded areas indicate the standard deviation.

hundred impressions, while DBGD or MGD continue to learn and considerably improve over the total 30,000 impressions. Because the pairwise baseline also converges sub-optimally under the perfect click model, we do not attribute its suboptimal convergence to noise but to the method being biased.

Conversely, Table 3 shows that PDGD reaches significantly higher performance than all the baselines within 10,000 impressions. Improvements are observed on all datasets under all levels of noise, especially on the commercial datasets where increases up to 0.17 NDCG are observed. Our results also show that PDGD learns faster than the baselines; at all time-steps the offline performance of PDGD is at least as good or better than all other methods, across all datasets. This increased learning speed can also be observed in Figure 2. Besides the faster learning it also appears as if PDGD converges at a better optimum than DBGD or MGD. However, Figure 2 reveals that DBGD does not fully converge within 30,000 iterations. Therefore, we performed an additional experiment where PDGD and DBGD optimize models over 1,000,000 impressions on the MSLR-WEB10k dataset, as displayed in Figure 3. Clearly the performance of DBGD plateaus at a considerably lower level than that of PDGD. Therefore, we conclude that PDGD indeed has an improved point of final convergence compared to DBGD and MGD.

Finally, Figure 2 and 3 also shows the behavior predicted by the speed-quality tradeoff [23]: a more complex model will have a worse initial performance but a better final convergence. Here, we see that depending on the level of interaction noise the neural model requires 3,000 to 20,000 iterations to match the performance of a linear model. However, in the long run the neural model does converge at a significantly better point of convergence. Thus, we conclude that PDGD is capable of effectively optimizing different kinds of models in terms of offline performance.

In conclusion, our results show that PDGD learns faster than existing OLTR methods while also converging at significantly better levels of performance.

5.2 User experience during training

Besides the ranking models learned by the OLTR methods, we also consider the user experience during optimization. Table 4 shows that the online performance of DBGD and MGD are close to each other; MGD has a higher online performance due to its faster learning speed [25, 33]. In contrast, the pairwise baseline has a substantially lower online performance in all cases. Because Figure 2 shows that the learning speed of the pairwise baseline sometimes matches that of DBGD and MGD, we attribute this difference to the exploration strategy it uses. Namely, the random insertion of uniformly sampled documents by this baseline appears to have a strong negative effect on the user experience.

The linear model optimized by PDGD has significant improvements over all baseline methods on all datasets and under all click models. This improvement indicates that the exploration of PDGD,

Table 4: Online performance (Discounted Cumulative NDCG, Section 4.4) for different instantiations of CCM (Table 2). The standard deviation is shown in brackets, bold values indicate the highest performance per dataset and click model, significant improvements and losses over the DBGD, MGD and pairwise baselines are indicated by △ ($p < 0.05$) and ▲ ($p < 0.01$) and by ▽ and ▼, respectively.

	MQ2007	**MQ2008**	**MSLR-WEB10k**	**Yahoo**	**istella**
			perfect		
DBGD (linear)	675.7 (21.8)	843.6 (40.8)	533.6 (15.6)	1159.3 (31.6)	589.9 (19.2)
DBGD (neural)	602.7 (58.1)	776.9 (67.4)	481.2 (53.0)	1135.7 (41.3)	494.3 (60.5)
MGD (linear)	689.6 (15.3)	858.6 (40.6)	558.7 (6.4)	1203.9 (9.9)	670.9 (8.6)
Pairwise (linear)	458.4 (13.3)	616.6 (25.8)	345.3 (4.6)	1027.2 (9.2)	64.5 (2.1)
PDGD (linear)	**797.3 (17.3)** ▲ ▲ ▲ ▲	**959.7 (43.4)** ▲ ▲ ▲ ▲	**691.4 (12.3)** ▲ ▲ ▲ ▲	**1360.3 (10.8)** ▲ ▲ ▲ ▲	**957.5 (9.4)** ▲ ▲ ▲ ▲
PDGD (neural)	743.7 (18.8) ▲ ▲ ▲ ▲	925.4 (43.3) ▲ ▲ ▲ ▲	619.2 (13.6) ▲ ▲ ▲ ▲	1319.6 (10.1) ▲ ▲ ▲ ▲	834.0 (22.2) ▲ ▲ ▲ ▲
			navigational		
DBGD (linear)	638.6 (29.7)	816.9 (42.0)	508.2 (21.6)	1129.9 (32.2)	538.2 (29.0)
DBGD (neural)	573.7 (68.4)	740.3 (69.7)	465.8 (52.0)	1116.0 (45.7)	414.3 (96.2)
MGD (linear)	635.9 (14.7)	824.5 (34.0)	538.1 (7.6)	1181.7 (20.0)	593.2 (9.7)
Pairwise (linear)	459.9 (12.9)	618.6 (25.2)	347.3 (5.4)	1031.2 (9.0)	72.6 (2.2)
PDGD (linear)	**703.0 (17.9)** ▲ ▲ ▲ ▲	**903.1 (40.7)** ▲ ▲ ▲ ▲	**578.1 (16.0)** ▲ ▲ ▲ ▲	**1298.4 (33.4)** ▲ ▲ ▲ ▲	**704.1 (33.5)** ▲ ▲ ▲ ▲
PDGD (neural)	560.9 (14.6) ▼ ▽ ▼ ▲	788.7 (38.5) ▼ ▲ ▼ ▲	448.1 (12.3) ▼ ▼ ▼ ▲	1176.1 (17.0) ▲ ▲ ▽ ▲	390.2 (35.1) ▼ ▼ ▼ ▲
			informational		
DBGD (linear)	584.2 (41.1)	757.4 (56.9)	477.2 (32.2)	1110.0 (37.0)	436.8 (57.4)
DBGD (neural)	550.8 (75.7)	720.9 (79.0)	444.7 (60.9)	1091.2 (48.6)	322.9 (121.0)
MGD (linear)	618.8 (21.7)	815.1 (44.5)	540.0 (7.7)	1159.1 (40.0)	581.8 (10.7)
Pairwise (linear)	462.6 (14.4)	619.6 (25.0)	349.7 (6.6)	1034.1 (9.0)	77.0 (2.4)
PDGD (linear)	**704.8 (30.5)** ▲ ▲ ▲ ▲	**907.9 (42.0)** ▲ ▲ ▲ ▲	**567.3 (36.5)** ▲ ▲ ▲ ▲	**1266.7 (50.0)** ▲ ▲ ▲ ▲	**731.5 (80.0)** ▲ ▲ ▲ ▲
PDGD (neural)	594.6 (23.0) △ ▲ ▼ ▲	818.3 (39.6) ▲ ▲ ▲	470.1 (19.4) ▽ ▲ ▼ ▲	1178.1 (22.8) ▲ ▲ ▲ ▲	484.3 (64.8) ▲ ▲ ▼ ▲

which uses a distribution over documents, does not lead to a worse user experience. In conclusion, PDGD provides a considerably better user experience than all existing methods.

Finally, we also discuss the performance of the neural models optimized by PDGD and DBGD. This model has both significant increases and decreases in online performance varying per dataset and amount of interaction noise. The decrease in user experience is predicted by the speed-quality tradeoff [23], as Figure 2 also shows, the neural model has a slower learning speed leading to a worse initial user experience. A solution to this tradeoff has been proposed by Oosterhuis and de Rijke [23], which optimizes a cascade of models. In this case, the cascade could combine the user experience of the linear model with the final convergence of the neural model, providing the best of both worlds.

5.3 Improvements of PDGD

After having discussed the offline and online performance of PDGD, we will now answer RQ1 and RQ3.

First, concerning RQ1 (whether PDGD performs significantly better than MGD), the results of our experiments show that models optimized with PDGD learn faster and converge at better optima than MGD, DBGD, and the pairwise baseline, regardless of dataset or level of interaction noise. Moreover, the level of performance reached with PDGD is significantly higher than the final convergence of any other method. Thus, even in the long run DBGD and MGD are incapable of reaching the offline performance of PDGD. Additionally, the online performance of a linear model optimized with PDGD is significantly better across all datasets and user models. Therefore, we answer RQ1 positively: PDGD outperforms existing methods both in terms of model convergence and user experience during learning.

Then, with regards to RQ3 (whether PDGD can effectively optimize different types of models), in our experiments we have successfully optimized models from two families: linear models and neural networks. Both models reach a significantly higher level of performance of model convergence than previous OLTR methods, across all datasets and degrees of interaction noise. As expected, the simpler linear model has a better initial user experience, while the more complex neural model has a better point of convergence. In conclusion, we answer RQ3 positively: PDGD is applicable to different ranking models and effective for both linear and non-linear models.

6 CONCLUSION

In this paper, we have introduced a novel OLTR method: PDGD that estimates its gradient using inferred pairwise document preferences. In contrast with previous OLTR approaches PDGD does not rely on online evaluation to update its model. Instead after each user interaction it infers preferences between document pairs. Subsequently, it constructs a pairwise gradient that updates the ranking model according to these preferences.

We have proven that this gradient is unbiased w.r.t. user preferences, that is, if there is a preference between a document pair, then in expectation the gradient will update the model to meet this preference. Furthermore, our experimental results show that PDGD learns faster and converges at a higher performance level than existing OLTR methods. Thus, it provides better performance in the short and long term, leading to an improved user experience during training as well. On top of that, PDGD is also applicable to any differentiable ranking model, in our experiments a linear and a neural network were optimized effectively. Both reached significant improvements over DBGD and MGD in performance at convergence. In conclusion, the novel unbiased PDGD algorithm provides better performance than existing methods in terms of convergence and user experience. Unlike the previous state-of-the-art, it can be applied to any differentiable ranking model.

Future research could consider the regret bounds of PDGD; these could give further insights into why it outperforms DBGD based methods. Furthermore, while we proved the unbiasedness of our method w.r.t. document pair preferences, the expected gradient weighs document pairs differently. Offline LTR methods like LambdaMART [2] use a weighted pairwise loss to create a listwise method that directly optimizes IR metrics. However, in the online setting there is no metric that is directly optimized. Instead, future work could see if different weighing approaches are more in line with user preferences. Another obvious avenue for future research is to explore the effectiveness of different ranking models in the online setting. There is a large collection of research in ranking models in offline LTR, with the introduction of PDGD such an extensive exploration in models is now also possible in OLTR.

Code

To facilitate reproducibility of the results in this paper, we are sharing the code used to run the experiments in this paper at https://github.com/HarrieO/OnlineLearningToRank.

Acknowledgements

This research was partially supported by the Netherlands Organisation for Scientific Research (NWO) under project nr. 612.001.551.

REFERENCES

[1] James Allan, Ben Carterette, Javed A Aslam, Virgil Pavlu, Blagovest Dachev, and Evangelos Kanoulas. 2007. Million query track 2007 overview. In *TREC*. NIST.
[2] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An overview.* Technical Report. Microsoft Research.
[3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise Approach to listwise approach. In *ICML*. ACM, 129–136.
[4] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research* 14 (2011), 1–24.
[5] Sergiu Chelaru, Claudia Orellana-Rodriguez, and Ismail Sengor Altingovde. 2014. How useful is social feedback for learning to rank YouTube videos? *World Wide Web* 17, 5 (2014), 997–1025.
[6] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search.* Morgan & Claypool Publishers.
[7] 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), 15.
[8] Susan T. Dumais. 2010. The web changes everything: Understanding and supporting people in dynamic information environments. In *ECDL*. Springer, 1.
[9] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 249–256.
[10] Fan Guo, Chao Liu, and Yi M Wang. 2009. Efficient multiple-click models in web search. In *WSDM*. ACM, 124–131.
[11] Jing He, Chengxiang Zhai, and Xiaoming Li. 2009. Evaluation of methods for relative comparison of retrieval systems based on clickthroughs. In *CIKM*. ACM, 2029–2032.
[12] Katja Hofmann. 2013. *Fast and Reliably Online Learning to Rank for Information Retrieval.* Ph.D. Dissertation. University of Amsterdam.
[13] Katja Hofmann, Anne Schuth, Shimon Whiteson, and Maarten de Rijke. 2013. Reusing historical interaction data for faster online learning to rank for IR. In *WSDM*. ACM, 183–192.
[14] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2011. Balancing Exploration and Exploitation in Learning to Rank Online. In *ECIR*. Springer, 251–263.
[15] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2011. A probabilistic method for inferring preferences from clicks. In *CIKM*. ACM, 249–258.
[16] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2012. Balancing Exploration and Exploitation in Listwise and Pairwise Online Learning to Rank for Information Retrieval. *Information Retrieval* 16, 1 (2012), 63–90.
[17] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *KDD*. ACM, 133–142.
[18] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. 2013. Learning to rank for recommender systems. In *RecSys*. ACM, 493–494.
[19] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In *SIGIR*. ACM, 475–484.
[20] Damien Lefortier, Pavel Serdyukov, and Maarten de Rijke. 2014. Online exploration for detecting shifts in fresh intent. In *CIKM*. ACM, 589–598.
[21] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.
[22] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. 2007. LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval. In *LR4IR '07*.
[23] Harrie Oosterhuis and Maarten de Rijke. 2017. Balancing speed and quality in online learning to rank for information retrieval. In *CIKM*. ACM, 277–286.
[24] Harrie Oosterhuis and Maarten de Rijke. 2017. Sensitive and scalable online evaluation with theoretical guarantees. In *CIKM*. ACM, 77–86.
[25] Harrie Oosterhuis, Anne Schuth, and Maarten de Rijke. 2016. Probabilistic Multileave Gradient Descent. In *ECIR*. Springer, 661–668.
[26] Art B Owen. 2013. Monte Carlo theory, methods and examples. *Monte Carlo Theory, Methods and Examples. Art Owen* (2013).
[27] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597* (2013).
[28] Filip Radlinski and Nick Craswell. 2017. A theoretical framework for conversational search. In *CHIIR*. 117–126.
[29] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. 2008. Learning diverse rankings with multi-armed bandits. In *ICML*. ACM Press, 784–791.
[30] Filip Radlinski, Madhu Kurup, and Thorsten Joachims. 2008. How does clickthrough data reflect retrieval quality?. In *CIKM*. ACM, 43–52.
[31] Mark Sanderson. 2010. Test Collection Based Evaluation of Information Retrieval Systems. *Foundations and Trends in Information Retrieval* 4, 4 (2010), 247–375.
[32] Anne Schuth, Robert-Jan Bruintjes, Fritjof Büttner, Joost van Doorn, and others. 2015. Probabilistic multileave for online retrieval evaluation. In *SIGIR*. ACM, 955–958.
[33] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. 2016. Multileave gradient descent for fast online learning to rank. In *WSDM*. ACM, 457–466.
[34] Anne Schuth, Floor Sietsma, Shimon Whiteson, Damien Lefortier, and Maarten de Rijke. 2014. Multileaved Comparisons for Fast Online Evaluation. In *CIKM*. ACM, 71–80.
[35] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. 2013. Ranked Bandits in Metric Spaces: Learning Diverse Rankings over Large Document Collections. *Journal of Machine Learning Research* 14, 1 (2013), 399–436.
[36] Balázs Szörényi, Róbert Busa-Fekete, Adil Paul, and Eyke Hüllermeier. 2015. Online rank elicitation for Plackett-Luce: A dueling bandits approach. In *NIPS*. 604–612.
[37] Pertti Vakkari and Nana Hakala. 2000. Changes in relevance criteria and problem stages in task performance. *Journal of Documentation* 56 (2000), 540–562.
[38] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *SIGIR*. ACM, 115–124.
[39] Yisong Yue and Thorsten Joachims. 2009. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML*. 1201–1208.
[40] Yisong Yue, Rajan Patel, and Hein Roehrig. 2010. Beyond position bias: Examining result attractiveness as a source of presentation bias in clickthrough data. In *WWW*. ACM, 1011–1018.
[41] Masrour Zoghi, Tomáš Tunys, Mohammad Ghavamzadeh, Branislav Kveton, Csaba Szepesvari, and Zheng Wen. 2017. Online Learning to Rank in Stochastic Click Models. In *ICML*. 4199–4208.
[42] Masrour Zoghi, Shimon Whiteson, Maarten de Rijke, and Remi Munos. 2014. Relative confidence sampling for efficient on-line ranker evaluation. In *WSDM*. 73–82.