Counterfactual Learning-to-Rank for Additive Metrics and Deep Models

Aman Agarwal
Cornell University
Ithaca, NY
aman@cs.cornell.edu

Ivan Zaitsev
Cornell University
Ithaca, NY
iz44@cornell.edu

Thorsten Joachims
Cornell University
Ithaca, NY
tj@cs.cornell.edu

ABSTRACT

Implicit feedback (e.g., clicks, dwell times) is an attractive source of training data for Learning-to-Rank, but it inevitably suffers from biases such as position bias. It was recently shown how counterfactual inference techniques can provide a rigorous approach for handling these biases, but existing methods are restricted to the special case of optimizing average rank for linear ranking functions. In this work, we generalize the counterfactual learning-to-rank approach to a broad class of additive rank metrics – like Discounted Cumulative Gain (DCG) and Precision@k – as well as non-linear deep network models. Focusing on DCG, this conceptual generalization gives rise to two new learning methods that both directly optimize an unbiased estimate of DCG despite the bias in the implicit feedback data. The first, SVM PropDCG, generalizes the Propensity Ranking SVM (SVM PropRank), and we show how the resulting optimization problem can be addressed via the Convex Concave Procedure (CCP). The second, Deep PropDCG, further generalizes the counterfactual learning-to-rank approach to deep networks as non-linear ranking functions. In addition to the theoretical support, we empirically find that SVM PropDCG significantly outperforms SVM PropRank in terms of DCG, and that it is robust to varying severity of presentation bias, noise, and propensity-model misspecification. Moreover, the ability to train non-linear ranking functions via Deep PropDCG further improves DCG.

KEYWORDS

Unbiased learning to rank, Discounted Cumulative Gain, counterfactual inference

ACM Reference Format:
Aman Agarwal, Ivan Zaitsev, and Thorsten Joachims. 2018. Counterfactual Learning-to-Rank for Additive Metrics and Deep Models. In *Proceedings of Preprint*. ACM, New York, NY, USA, 10 pages. https://doi.org/xx

1 INTRODUCTION

Implicit feedback from user behavior is an attractive source of data in many information retrieval (IR) systems, especially ranking applications where collecting relevance annotations from experts can be economically infeasible or even impossible (e.g., personal collection search, intranet search, scholarly search). While implicit feedback is often abundant, cheap, timely, user-centric, and routinely logged, it suffers from inherent biases. For example, the position of a result in a search ranking strongly affects how likely it is to be seen by a user and thus clicked. So, naively using click data as a relevance signal leads to sub-optimal performance.

A counterfactual inference approach for learning-to-rank (LTR) from logged implicit feedback was recently developed to deal with such biases [11]. This method provides a rigorous approach to unbiased learning despite biased data and overcomes the limitations of alternative bias-mitigation strategies. In particular, it does not require the same query to be seen multiple times as necessary for most generative click models, and it does not introduce alternate biases like treating clicks as preferences between clicked and skipped documents.

The key technique in counterfactual learning is to incorporate the propensity of obtaining a particular training example into an Empirical Risk Minimization (ERM) objective that is provably unbiased [21]. While it was shown that this is possible for learning to rank, existing work is limited to linear ranking functions and optimizing average rank of the relevant documents as objective [11]. In this paper, we generalize the counterfactual LTR framework to a broad class of additive IR metrics as well as non-linear deep models. Specifically, we show that any IR metric that is the sum of individual document relevances weighted by some function of document rank can be directly optimized via Propensity-Weighted Empirical Risk Minimization. Moreover, if an IR metric meets the mild requirement that the rank weighting function is monotone, then a hinge-loss upper bounding technique allows learning linear ranking functions via a Ranking SVM, as well as learning non-linear ranking functions via deep networks.

With the general framework in place, we fully develop two learning-to-rank methods that optimize the Discounted Cumulative Gain (DCG) metric. The first is SVM PropDCG, which generalizes a Ranking SVM to directly optimize a bound on DCG from biased click data. The resulting optimization problem is no longer convex, and we show how to find a local optimum using the Convex Concave Procedure (CCP). In CCP, several iterations of convex subproblems are solved. In the case of SVM PropDCG, these convex sub-problems have the convenient property of being a Quadratic Program analogous to a generalized Ranking SVM. This allows the CCP to work by invoking an existing and fast SVM solver in each iteration until convergence. The second method we develop is Deep PropDCG, which further generalizes the approach to deep networks as non-linear ranking functions. Deep PropDCG also optimizes a bound on the DCG, and we show how the resulting optimization problem can be solved via stochastic gradient descent for any network architecture that shares neural network weights

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Preprint, 2018,
© 2018 Copyright held by the owner/author(s).
ACM ISBN xx…$15.00
https://doi.org/xx

across candidate documents for the same query.

In addition to the theoretical derivation and the justification it provides, we also empirically evaluate the effectiveness of both SVM PropDCG and Deep PropDCG, especially in comparison to the existing SVM PropRank method [11]. We find that SVM PropDCG performs significantly better than SVM PropRank in terms of DCG, and that it is robust to varying degrees of bias, noise and propensity-model misspecification. In our experiments, CCP convergence was typically achieved quickly within three to five iterations. For Deep PropDCG, the results show that DCG performance is further improved compared to SVM PropDCG when using a two-layer neural network, thus demonstrating that the counterfactual learning approach can effectively train non-linear ranking functions. The software for both SVM PropDCG and Deep PropDCG will be made available by the time of the conference.

2 RELATED WORK

Generative click models are a popular approach for explaining the bias in user behavior and for extracting relevance labels for learning. For example, in the cascade model [7] users are assumed to sequentially go down a ranking and click on a document, thus revealing preferences between clicked and skipped documents. Learning from these relative preferences lowers the impact of some biases [9]. Other click models ([1, 5, 7], also see [6]) train to maximize log-likelihood of observed clicks, where relevance is modeled as a latent variable that is inferred over multiple instances of the same query. In contrast, the counterfactual framework [11] does not require latent-variable inference and repeat queries, but allows directly incorporating click feedback into the learning-to-rank algorithm in a principled and unbiased way, thus allowing the direct optimization of ranking performance over the natural query distribution.

The counterfactual approach uses inverse propensity scoring (IPS), originally employed in survey sampling [8] and causal inference from observational studies [17], but more recently also in whole page optimization [27], IR evaluation with manual judgments [18], and recommender evaluation [12, 19]. This approach is similar in spirit to [25], where propensity-weighting is used to correct for selection bias when discarding queries without clicks during learning-to-rank.

While our focus is on directly optimizing ranking performance in the implicit feedback partial-information setting, several approaches have been proposed for the same task in the full-information supervised setting, i.e. when the relevances of all the documents in the training set are known. A common strategy is to use some smoothed version of the ranking metric for optimization, as seen in SoftRank [24] and others [4, 10, 28, 29]. In particular, SoftRank optimizes the expected performance metric over the distribution of rankings induced by smoothed scores, which come from a normal distribution centered at the query-document mean scores predicted by a neural net. This procedure is computationally expensive with an $O(n^3)$ dependence on the number of documents for a query. In contrast, our approach employs an upper bound on the performance metric, whose structure makes it amenable to the Convex Concave Procedure for efficient optimization, as well as adaptable to non-linear ranking functions via deep networks.

Finally, several works exist [2, 3, 16, 24] that have proposed neural network architectures for learning-to-rank. We do not focus on a specific network architecture in this paper, but instead propose a new training criterion for learning-to-rank from implicit feedback that in principle allows unbiased network training for a large class of architectures.

Metric	$\lambda(r)$
AvgRank	r
DCG	$-1/\log(1+r)$
Prec@k	$-\mathbf{1}_{r\leq k}/k$
RBP-p [15]	$-(1-p)/p^r$

Table 1: Some popular linearly decomposable IR metrics that can be directly optimized by Propensity-Weighted ERM. $\lambda(r)$ is the rank weighting function.

3 UNBIASED LEARNING FOR RANK-BASED IR METRICS

We begin by generalizing the counterfactual learning framework from [11] to the class of linearly decomposable metrics as defined below. Suppose we are given a sample $\boldsymbol{X}$ of i.i.d. query instances $\boldsymbol{x}_i \sim \mathrm{P}(\boldsymbol{x})$, $i \in [N]$. A query instance can include personalized and contextual information about the user in addition to the query string. For each query instance $\boldsymbol{x}_i$, let $\mathrm{r}_i(y)$ denote the user-specific relevance of result y for instance $\boldsymbol{x}_i$. For simplicity, assume that relevances are binary, $\mathrm{r}_i(y) \in \{0,1\}$. In the following, we consider the class of additive ranking performance metrics, which includes any metric that can be expressed as

$$\Delta(\boldsymbol{y}|\boldsymbol{x}_i, \mathrm{r}_i) \quad = \quad \sum_{y \in \boldsymbol{y}} \lambda\left(\mathrm{rank}(y|\boldsymbol{y})\right) \cdot \mathrm{r}_i(y). \tag{1}$$

$\boldsymbol{y}$ denotes a ranking of results, and $\lambda()$ can be any weighting function that depends on the rank $\mathrm{rank}(y|\boldsymbol{y})$ of document y in ranking $\boldsymbol{y}$. A broad range of commonly used ranking metrics falls into this class, and Table 1 lists some of them. For instance, setting $\lambda(r) = r$ gives the sum of relevant ranks metric (also called average rank when normalized) considered in [11], and $\lambda(r) = \frac{-1}{\log(1+r)}$ gives the DCG metric. Note that we consider negative values wherever necessary to make the notation consistent with risk minimization.

A ranking system S maps a query instance $\boldsymbol{x}_i$ to a ranking $\boldsymbol{y}$. Aggregating the losses of individual rankings over the query distribution, we can define the overall *risk* (e.g., the expected DCG) of a system as

$$R(S) \quad = \quad \int \Delta(S(\boldsymbol{x})|\boldsymbol{x}, \mathrm{r}) \, d\,\mathrm{P}(\boldsymbol{x}, \mathrm{r}). \tag{2}$$

A key problem when working with implicit feedback data is that we cannot assume that all relevances r_i are observed. In particular, while a click (or a sufficiently long dwell time) provides a noisy indicator of positive relevance in the presented ranking $\bar{\boldsymbol{y}}_i$, a missing click does not necessarily indicate lack of relevance as the user may not have observed that result. From a machine learning perspective, this implies that we are in a partial-information setting, which we will deal with by explicitly modeling missingness in addition to relevance. Let $o_i \sim \mathrm{P}(o|\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i, \mathrm{r}_i)$ denote the 0/1 vector indicating which relevance values are revealed. While o_i is not necessarily fully observed either, we can now model its distribution, which we will find below is sufficient for unbiased learning despite the missing data. In particular, the *propensity* of observing $\mathrm{r}_i(y)$ for query instance x_i given presented ranking $\bar{\boldsymbol{y}}$ is then defined as $Q(o_i(y)=1|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,\mathrm{r}_i)$.

Using this counterfactual setup, an unbiased estimate of $\Delta(\boldsymbol{y}|\boldsymbol{x}_i,\mathrm{r}_i)$ for any ranking $\boldsymbol{y}$ can be obtained via inverse propensity scoring

$$\hat{\Delta}_{IPS}(\boldsymbol{y}|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,o_i) = \sum_{\substack{y:o_i(y)=1 \\ \wedge\, \mathrm{r}_i(y)=1}} \frac{\lambda\left(\mathrm{rank}(y|\boldsymbol{y})\right)}{Q(o_i(y)=1|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,\mathrm{r}_i)}. \tag{3}$$

This is an unbiased estimate if $Q(o_i(y)=1|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,\mathrm{r}_i) > 0$ for all y that are relevant $\mathrm{r}_i(y)=1$. The proof follows directly from [11] with $\mathrm{rank}(y|\boldsymbol{y})$ replaced with $\lambda\left(\mathrm{rank}(y|\boldsymbol{y})\right)$. Note that the estimator in Equation (3) sums only over the results where the feedback is observed (i.e., $o_i(y)=1$) and positive (i.e., $\mathrm{r}_i(y)=1$), which means that we do not have to disambiguate whether lack of positive feedback (e.g., the lack of a click) is due to a lack of relevance or due to missing the observation (e.g., result not relevant vs. not viewed).

Using this unbiased estimate of the loss function, we get an unbiased estimate of the risk of a system

$$\hat{R}_{IPS}(S) = \frac{1}{N}\sum_{i=1}^{N} \sum_{\substack{y:o_i(y)=1 \\ \wedge\, \mathrm{r}_i(y)=1}} \frac{\lambda\left(\mathrm{rank}(y|S(\boldsymbol{x}_i))\right)}{Q(o_i(y)=1|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,\mathrm{r}_i)}. \tag{4}$$

This propensity-weighted empirical risk can be used to perform Empirical Risk Minimization (ERM)

$$\hat{S} = \mathrm{argmin}_{S\in\mathcal{S}} \left\{\hat{R}_{IPS}(S)\right\}.$$

Again, the justification for consistency provided in [11] holds for general metrics. So intuitively, given enough training data, the learning algorithm is guaranteed to find the best system in $\mathcal{S}$.

3.1 Propensity Model

Search engine click logs provide a sample of query instances $\boldsymbol{x}_i$, the presented ranking $\bar{\boldsymbol{y}}_i$ and a (sparse) click-vector where each $\mathrm{c}_i(y) \in \{0,1\}$ indicates whether result y was clicked or not. To compute the propensity of a click, one can employ click models that distinguish between examination and relevance. A simple model for how a user examines ($\mathrm{e}_i(y)$) and then clicks a result y is the position-based model (PBM)

$$P(\mathrm{e}_i(y)=1|\,\mathrm{rank}(y|\bar{\boldsymbol{y}}))\cdot P(\mathrm{c}_i(y)=1|\,\mathrm{r}_i(y),\mathrm{e}_i(y)=1).$$

In the PBM, examination depends only on the rank of a result in the presented ranking. If we make a further simplifying assumption that a user clicks if and only if the result is relevant and examined, then examination equals observation for relevant results, i.e. $Q(o_i(y)|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,\mathrm{r}_i=1) \equiv P(\mathrm{e}_i(y)|\,\mathrm{rank}(y|\bar{\boldsymbol{y}}_i)) \equiv p_{\mathrm{rank}(y|\bar{\boldsymbol{y}}_i)}$. This simplifies the risk estimate from (4) to

$$\hat{R}_{IPS}(S) = \frac{1}{n}\sum_{i=1}^{n} \sum_{y:\mathrm{c}_i(y)=1} \frac{\lambda\left(\mathrm{rank}(y|S(\boldsymbol{x}_i))\right)}{p_{\mathrm{rank}(y|\bar{\boldsymbol{y}}_i)}}, \tag{5}$$

where the sum is over the clicked documents.

The propensities p_r can be estimated by conducting a small swap-intervention experiment [11] which swaps results at some "landmark" rank k and all other ranks r in individual interventions. Alternatively, the approach in [26] can be used to estimate the propensity model from observational data via Expectation-Maximization without any online intervention.

The position-based model is arguably the simplest propensity model for counterfactual learning-to-rank, and we conjecture that more sophisticated models may provide better results. Fortunately, the learning-to-rank methods we develop in the following are agnostic to the propensity model that is used as they merely require the computed propensity value $q_i \in [0,1]$ as input. Thus, changes in the propensity model do not require changes to the learning methods.

3.2 Incorporating Click Noise

In Section 3.1, clicks were assumed to reveal relevance without noise. This restriction is not necessary, and let's consider noisy clicks on both the relevant and the non-relevant results with $1 \geq \epsilon_+ > \epsilon_- \geq 0$ and

$$P(\mathrm{c}_i(y)=1|\,\mathrm{r}_i(y)=1,\mathrm{e}_i(y)=1) = \epsilon_+,$$
$$P(\mathrm{c}_i(y)=1|\,\mathrm{r}_i(y)=0,\mathrm{e}_i(y)=1) = \epsilon_-.$$

Note that setting $\epsilon_+ = 1$ and $\epsilon_- = 0$ reduces to the noise-free analysis of the previous section.

The ERM approach is valid even under this noise model. This is because in expectation the noise is order-preserving for $R(S)$, so the minimizer of (4) asymptotically remains the same despite noisy clicks. In fact, the proof in [11] generalizes directly for all additive IR metrics.

4 LEARNING METHODS

The previous section provides a theoretically justified training objective for learning-to-rank with additive metrics like DCG. However, it remains to be shown that this training objective can be implemented in efficient and practical learning methods. This section shows that this is indeed possible for a generalization of Ranking SVMs and for deep networks as ranking functions.

Consider a dataset of n examples of the following form. For each query-result pair $(\boldsymbol{x}_i,y_i)$ that is clicked, let $q_i = Q(o_i(y)=1|\boldsymbol{x}_i,\bar{\boldsymbol{y}}_i,\mathrm{r}_i)$ be the propensity of the click according to a click propensity model. We also record the candidate set Y_i of all results for query $\boldsymbol{x}_i$. Note that each click generates a separate training example, even if multiple clicks occur for the same query.

Given this propensity-scored click data, we would like to learn a scoring function $f(\boldsymbol{x},y)$. Such a scoring function f naturally specifies a ranking system S by sorting candidate results Y for a given query $\boldsymbol{x}$ by their scores.

$$S_f(\boldsymbol{x}) \equiv \mathrm{argsort}_Y \left\{f(\boldsymbol{x},y)\right\} \tag{6}$$

Since $\mathrm{rank}(y|S_f(\boldsymbol{x}))$ of a result is a discontinuous step function of the score, tractable learning algorithms typically optimize a substitute loss that is (sub-)differentiable [9, 24, 29]. Following this route, we now derive a tractable substitute for the empirical risk of (4) in terms of the scoring function. This is achieved by the following hinge-loss upper bound [11] on the rank

$$\begin{aligned}\mathrm{rank}(y_i|\boldsymbol{y}) - 1 &= \sum_{\substack{y \in Y_i \\ y \neq y_i}} \mathbb{1}_{f(\boldsymbol{x}_i, y) - f(\boldsymbol{x}_i, y_i) > 0} \\ &\leq \sum_{\substack{y \in Y_i \\ y \neq y_i}} \max(1 - (f(\boldsymbol{x}_i, y_i) - f(\boldsymbol{x}_i, y)), 0)\end{aligned}$$

Using this upper bound, we can also get a bound for any IR metric that can be expressed through a monotonically increasing weighting function $\lambda(r)$ of the rank. By rearranging terms and applying the weighting function $\lambda(r)$, we have

$$\lambda(\mathrm{rank}(y_i|\boldsymbol{y})) \leq \lambda\left(1 + \sum_{\substack{y \in Y_i \\ y \neq y_i}} \max(1 - (f(\boldsymbol{x}_i, y_i) - f(\boldsymbol{x}_i, y)), 0)\right).$$

This provides the following continuous and subdifferentiable upper bound $\hat{R}^{hinge}_{IPS}(f)$ on the propensity-weighted risk estimate of (4).

$$\begin{aligned}\hat{R}_{IPS}(S_f) &\leq \hat{R}^{hinge}_{IPS}(f) \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{q_i}\lambda\left(1 + \sum_{\substack{y \in Y_i \\ y \neq y_i}} \max(1 - (f(\boldsymbol{x}_i, y_i) - f(\boldsymbol{x}_i, y)), 0)\right) \quad (7)\end{aligned}$$

Focusing on the DCG metric, we show in the following how this upper bound can be optimized for linear as well as non-linear neural network scoring functions. For the general class of additive IR metrics, the optimization depends on the properties of the weighting function $\lambda(r)$, and we highlight them wherever appropriate. Note that the monotonicity condition is satisfied by all the metrics in Table 1.

4.1 SVM PropDCG

The following derives an SVM-style method, called SVM PropDCG, for learning a linear scoring function $f(\boldsymbol{x}, y) = w \cdot \phi(\boldsymbol{x}, y)$, where w is a weight vector and $\phi(\boldsymbol{x}, y)$ is a feature vector describing the match between query $\boldsymbol{x}$ and result y. For such linear ranking functions – which are widely used in Ranking SVMs [9] and many other learning-to-rank methods [14] –, the propensity-weighted ERM bound from Equation (7) can be expressed as the following SVM-type optimization problem.

$$\begin{aligned}\hat{w} \;&=\; \mathrm{argmin}_{w,\xi}\ \frac{1}{2} w \cdot w + \frac{C}{n}\sum_{i=1}^{n}\frac{1}{q_i}\lambda\left(\sum_{y \in Y_i} \xi_{iy} + 1\right) \\ s.t. \;\; & \forall y \in Y_1 \backslash \{y_1\} : w \cdot [\phi(\boldsymbol{x}_1, y_1) - \phi(\boldsymbol{x}_1, y)] \geq 1 - \xi_{1y} \\ & \vdots \\ & \forall y \in Y_n \backslash \{y_n\} : w \cdot [\phi(\boldsymbol{x}_n, y_n) - \phi(\boldsymbol{x}_n, y)] \geq 1 - \xi_{ny} \\ & \forall i \forall y : \xi_{iy} \geq 0\end{aligned}$$

C is a regularization parameter. The training objective optimizes the $\mathcal{L}_2$-regularized hinge-loss upper bound on the empirical risk estimate (7). This is because for any feasible (w, ξ) and monotone increasing weighting function $\lambda(r)$,

$$\begin{aligned}&\lambda\left(1 + \sum_{\substack{y \in Y_i \\ y \neq y_i}} \max(1 - (f(\boldsymbol{x}_i, y_i) - f(\boldsymbol{x}_i, y)), 0)\right) \\ &= \lambda\left(1 + \sum_{\substack{y \in Y_i \\ y \neq y_i}} \max(1 - w \cdot [\phi(\boldsymbol{x}_i, y_i) - \phi(\boldsymbol{x}_i, y)], 0)\right) \leq \lambda\left(1 + \sum_{y \in Y_i} \xi_{iy}\right)\end{aligned}$$

As shown in [11], for the special case of using the sum of relevant ranks as the metric to optimize, i.e. $\lambda(r) = r$, this SVM optimization problem is a convex Quadratic Program which can be solved efficiently using standard SVM solvers, like SVM-rank [10], via a one-slack formulation.

Moving to the case of DCG as the training metric via the weighting function $\lambda(r) = \frac{-1}{\log(1+r)}$, we get the following optimization problem for SVM PropDCG

$$\begin{aligned}\hat{w} \;&=\; \mathrm{argmin}_{w,\xi}\ \frac{1}{2} w \cdot w - \frac{C}{n}\sum_{i=1}^{n}\frac{1}{q_i}\frac{1}{\log(\sum_{y \in Y_i} \xi_{iy} + 2)} \\ s.t. \;\; & \forall j \forall y \in Y_i \backslash \{y_i\} : w \cdot [\phi(\boldsymbol{x}_i, y_i) - \phi(\boldsymbol{x}_i, y)] \geq 1 - \xi_{iy} \\ & \forall j \forall y : \xi_{iy} \geq 0.\end{aligned}$$

This optimization problem is no longer a convex Quadratic Program. However, all constraints are still linear inequalities in the variables w and ξ, and the the objective can be expressed as the difference of two convex functions. Let $h(w) = \frac{1}{2}\|w\|^2$ and $g(\xi) = \frac{C}{n}\sum_{j=1}^{n}\frac{1}{q_i}\frac{1}{\log(\sum_{y \in \mathrm{Y}_i} \xi_{iy} + 2)}$. Then the function h is the $\mathcal{L}_2$ norm of the vector w and is thus a convex function. As for the function g, the function $k : x \mapsto \frac{1}{\log x}$ is convex as it is the composition of a a convex decreasing function ($x \mapsto \frac{1}{x}$) with a concave function ($x \mapsto \log x$). So, since the sum of affine transformations of a convex function is convex, g is convex.

Such an optimization problem is called a convex-concave problem[1] and a local optimum can be obtained efficiently via the Convex-Concave Procedure (CCP) [13]. At a high level, the procedure works by repeatedly approximating the second convex function with its first order Taylor expansion which makes the optimization problem convex in each iteration. The Taylor expansion is first done at some chosen initial point in the feasible region, and then the solution of the convex problem in a particular iteration is used as the Taylor approximation point for the next iteration. It can be shown that this procedure converges to a local optimum [13].

[1] More generally, the inequality constraints can also be convex-concave and not just convex

Concretely, let w^k, ξ^k be the solution in the k^{th} iteration. Then, we have the Taylor approximation

$$\begin{aligned}\hat{g}(\xi;\xi^k) &= g(\xi^k) + \nabla g(\xi^k)^T(\xi - \xi^k)\\ = \quad & g(\xi^k) - \frac{C}{n}\sum_{j=1}^{n}\frac{1}{q_i}\frac{\sum_{y\in Y_i}\xi_{iy} - \xi_{iy}^k}{\left(\sum_{y\in Y_i}\xi_{iy}^k + 2\right)\log^2\left(\sum_{y\in Y_i}\xi_{iy}^k + 2\right)}\end{aligned}$$

Letting $q_i' = q_i\left(\sum_{y\in Y_i}\xi_{iy}^k + 2\right)\log^2\left(\sum_{y\in Y_i}\xi_{iy}^k + 2\right)$, and dropping the additive constant terms from $\hat{g}$, we get the following convex program that needs to be solved in each CCP iteration.

$$\begin{aligned}&\operatorname{argmin}_{w,\xi}\ \frac{1}{2}w\cdot w + \frac{C}{n}\sum_{i=1}^{n}\frac{1}{q_i'}\sum_{y\in Y_i}\xi_{iy}\\ s.t.\quad &\forall i\forall y\in Y_i\setminus\{y_i\}: w\cdot[\phi(\boldsymbol{x}_i,y_i) - \phi(\boldsymbol{x}_i,y)] \geq 1-\xi_{iy}\\ &\forall i\forall y:\xi_{iy}\geq 0\end{aligned}$$

Observe that this problem is of the same form as SVM PropRank, the Propensity Ranking SVM for the average rank metric, i.e. $\lambda(r) = r$ (with the caveat that q_i' are not propensities). This nifty feature allows us to solve the convex problem in each iteration of the CCP using the fast solver for SVM PropRank provided in [11]. In our experiments, CCP convergence was achieved within a few iterations – as detailed in the empirical section. For other IR metrics, the complexity and feasibility of the above Ranking SVM optimization procedure will depend on the form of the target IR metric. In particular, if the rank weighting function $\lambda(r)$ is convex, it may be solved directly as a convex program. If $\lambda(r)$ is concave, then the CCP may be employed as shown for the DCG metric above.

An attractive theoretical property of SVM-style methods is the ability to switch from linear to non-linear functions via the Kernel trick. In principle, kernelization can be applied to SVM PropDCG as is evident from the representer theorem [20]. Specifically, by taking the Lagrange dual, the problem can be kernelized analogous to [9]. While it can be shown that the dual is convex and strong duality holds, it is not clear that the optimization problem has a convenient and compact form that can be efficiently solved in practice. Even for the special case of the average rank metric, $\lambda(r) = r$, the associated kernel matrix $K_{iy,jy'}$ has a size equal to the total number of candidates $\sum_{i=1}^{n}|Y_i|$ squared, making the kernelization approach computationally infeasible or challenging at best. We therefore explore a different route for extending our approach to non-linear scoring functions in the following.

4.2 Deep PropDCG

Since moving to non-linear ranking functions through SVM kernelization is challenging, we instead explore deep networks as a class of non-linear scoring functions. Specifically, we replace the linear scoring function $f(\boldsymbol{x},y) = w\cdot\phi(\boldsymbol{x},y)$ with a neural network

$$f(\boldsymbol{x},y) = NN_w[\phi(\boldsymbol{x},y)] \tag{8}$$

This network is generally non-linear in both the weights w and the features $\phi(\boldsymbol{x},y)$. However, this does not affect the validity of the hinge-loss upper bound from Equation (7), which now takes the form

$$\frac{1}{n}\sum_{j=1}^{n}\frac{1}{q_i}\lambda\left(1+\sum_{\substack{y\in Y_i\\ y\neq y_i}}\max(1-(NN_w[\phi(\boldsymbol{x}_i,y_i)] - NN_w[\phi(\boldsymbol{x}_i,y)]),0)\right)$$

During training, we need to minimize this function with respect to the network parameters w. Unlike in the case of SVM PropDCG, this function can no longer be expressed as the difference of a convex and a concave function, since $NN_w[\phi(\boldsymbol{x}_i,y_i)]$ is neither convex nor concave in general. Nevertheless, the empirical success of optimizing non-convex $NN_w[\phi(\boldsymbol{x}_i,y_i)]$ via gradient descent to a local optimum is well documented, and we will use this approach in the following. This is possible since the training objective is subdifferentiable as long as the weighting function $\lambda(r)$ is differentiable. However, the non-linearity of $\lambda(r)$ adds a challenge in applying *stochastic* gradient descent methods to our training objective, since the objective no longer decomposes into a sum over all $(\boldsymbol{x}_i,y)$ as in standard network training. We discuss in the following how to handle this situation to arrive at an efficient stochastic-gradient procedure.

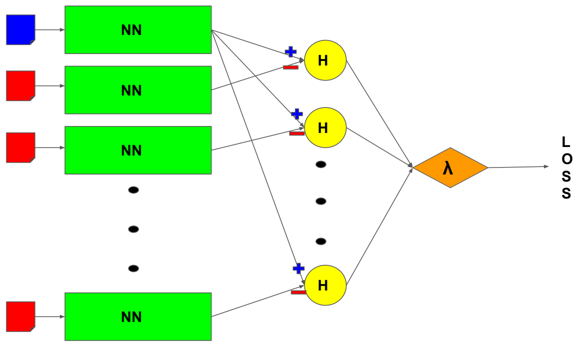

Figure 1: Deep PropDCG schema for computing the loss from one query instance. The blue document is the positive (clicked) result, and the red documents are the other candidates. The neural net NN is used to compute document scores for each set of candidate features. Pairs of scores are passed through the hinge node, and then finally the weighting function is applied as shown.

For concreteness, we again focus on the case of optimizing DCG via $\lambda(r) = \frac{-1}{\log(1+r)}$. In particular, plugging in the weighting function for DCG, we get the Deep PropDCG minimization objective

$$\frac{1}{n}\sum_{j=1}^{n}\frac{-1}{q_i}\log^{-1}\left(2+\sum_{\substack{y\in Y_i\\ y\neq y_i}}\max(1-(NN_w[\phi(\boldsymbol{x}_i,y_i)]-NN_w[\phi(\boldsymbol{x}_i,y)]),0)\right)$$

to which a regularization term can be added (our implementation uses weight decay).

Since the weighting function ties together the hinge losses from pairs of documents in a non-linear way, stochastic gradient descent (SGD) is not directly feasible at the level of individual documents. In the case of DCG, since the rank weighting function is concave, one possible workaround is a Majorization-Minimization scheme [22] (akin to CCP): upper bound the loss function with a linear Taylor approximation at the current neural net weights, perform SGD at the level of document pairs (y_i, y) to update the weights, and repeat until convergence.

While this Majorization-Minimization scheme in analogy to the SVM approach is possible also for deep networks, we chose a different approach for the reasons given below. In particular, given the success of stochastic-gradient training of deep networks in other settings, we directly perform stochastic-gradient updates at the level of query instances, not individual $(\boldsymbol{x}_i, y)$. At the level of query instances, the objective does decompose linearly such that any subsample of query instances can provide an unbiased gradient estimate. Note that this approach works for any differentiable weighting function $\lambda(r)$, does not require any alternating approximations as in Majorization-Minimization, and processes each candidate document y including the clicked document y_i only once in one SGD step.

For SGD at the level of query instances, a forward pass of the neural network – with the current weights fixed – must be performed on each document y in candidate set Y_i in order to compute the loss from training instance $(\boldsymbol{x}_i, y_i)$. Since the number of documents in each candidate set varies, this is best achieved by processing each input instance (including the corresponding candidate set) as a (variable-length) sequence so that the neural net weights are effectively shared across candidate documents for the same query instance.

This process is most easily understood via the network architecture illustrated in Figure 1. The scoring function $NN_w[\phi(\boldsymbol{x}_i, y_i)]$ is replicated for each result in the candidate set using shared weights w. In addition there is a hinge-loss node $H(u, v) = \max(1-(u-v), 0)$ that combines the score of the clicked result with each other result in the candidate set Y_i. For each such pair (y_i, y), the corresponding hinge-loss node computes its contribution h_j to the upper bound on the rank. The result of the hinge-loss nodes then feeds into a single weighting node $\Lambda(\vec{h}) = \lambda\left(1 + \sum_j h_j\right)$ that computes the overall bound on the rank and applies the weighting function. The result is the loss of that particular query instance.

Note that we have outlined a very general method which is agnostic about the size and architecture of the neural network. As a proof-of-concept, we achieved superior empirical results over a linear scoring function even with a simple two layer neural network, as seen in Section 5.7. We conjecture that DCG performance may be enhanced further with deeper, more specialized networks. Moreover, in principle, the hinge-loss nodes can be replaced with nodes that compute any other differentiable loss function that provides an upper bound on the rank without fundamental changes to the SGD algorithm.

5 EMPIRICAL EVALUATION

While the derivation of SVM PropDCG and Deep PropDCG has provided a theoretical justification for both methods, it still remains to show whether this theoretical argument translates to improved empirical performance. To this effect, the following empirical evaluation addresses three key questions.

First, we investigate whether directly optimizing DCG improve performance as compared to baseline methods, in particular SVM PropRank as the most relevant method for unbiased LTR from implicit feedback. Comparing SVM PropDCG to SVM PropRank is particularly revealing about the importance of direct DCG optimization, since both methods are linear SVMs and employ the same software machinery for the Quadratic Programs involved, thus eliminating any confounding factors. We also experimentally analyze the CCP optimization procedure to see whether SVM PropDCG is practical and efficient. Second, we explore the robustness of the generalized counterfactual LTR approach to noisy feedback, the severity of the presentation bias, and misspecification of the propensity model. And, finally, we compare the DCG performance of Deep PropDCG with a simple two layer neural network against the linear SVM PropDCG to understand to what extent non-linear models can be trained effectively using the generalized counterfactual LTR approach.

5.1 Setup

We conducted experiments on synthetic click data derived from the Yahoo Learning to Rank Challenge corpus. Our experiment setup is

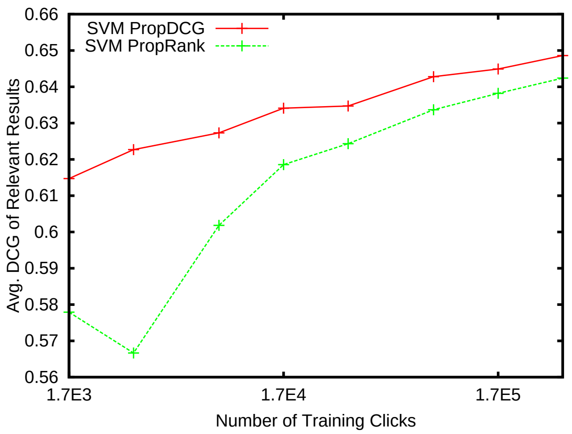

Figure 2: Test set Avg DCG performance for SVM PropDCG and SVM PropRank ($\eta = 1$, $\epsilon_- = 0.1$)

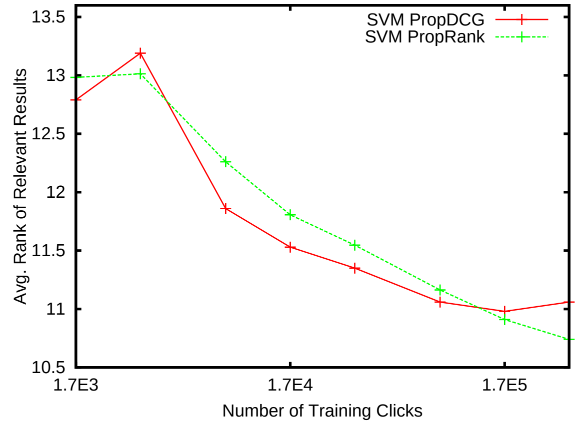

Figure 3: Test set Avg Rank performance for SVM PropDCG and SVM PropRank ($\eta = 1$, $\epsilon_- = 0.1$)

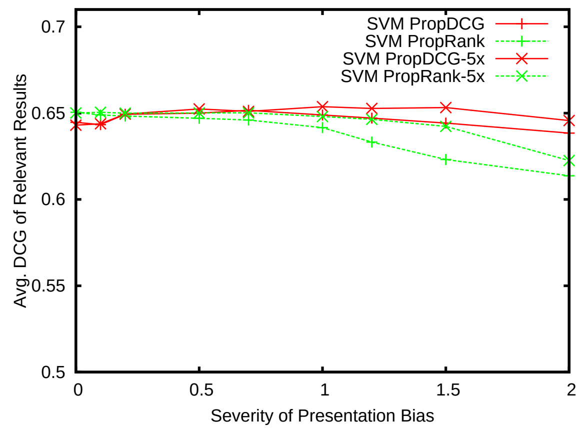

Figure 4: Test set Avg DCG performance for SVM PropDCG and SVM PropRank as presentation bias becomes more severe in terms of η ($n = 45K$ and $n = 225K$, $\epsilon_- = 0$).

borrowed directly from [11], and further details can be found there. Briefly, the training and validation click data were generated from the respective Yahoo full-information datasets (with relevances binarized) by simulating the position-based click model. Following [11], we use propensities that decay with presented rank of the result as $p_r = \left(\frac{1}{r}\right)^\eta$. The rankings that generate the clicks are given by a "production ranker" which was a conventional Ranking SVM trained on 1 percent of the full-information training data. The parameter η controls the severity of bias, with higher values causing greater position bias.

We also introduced noise into the clicks by allowing some irrelevant documents to be clicked. Specifically, an irrelevant document ranked at position r by the production ranker is clicked with probability p_r times ϵ_-. When not mentioned otherwise, we used the parameters $\eta = 1$, $\epsilon_- = 0.1$ and $\epsilon_+ = 1$, which is consistent with the setup used in [11]. Other bias profiles are also explored in the following.

Both the SVM PropRank and SVM PropDCG models were trained and cross-validated to pick the regularization constant C. For cross-validation, we use the partial feedback data in the validation set and select based on the IPS estimate of the DCG [23]. The performance of the models is reported on the binarized fully labeled test set which is never used for training or validation. While our target metric is DCG, we also report the average rank metric which is directly optimized in SVM PropRank for the sake of completeness.

5.2 How does ranking performance scale with training set size?

We first explore how the test-set ranking performance changes as the learning algorithm is given more and more click data. The resulting learning curves are given in Figures 2 and 3. The click data has presentation bias with $\eta = 1$ and noise with $\epsilon_- = 0.1$. For small datasets, results are averaged over 3 draws of the click data.

As expected, Figure 2 shows that the performance of both SVM PropDCG and SVM PropRank improves with increasing amounts of click data. Moreover, SVM PropDCG performs substantially better than the baseline SVM PropRank in maximizing test set DCG.

More surprisingly, both methods perform comparably in minimizing the average rank metric, with SVM PropDCG slightly better at smaller amounts of data and SVM PropRank better at larger amounts (Figure 3). We conjecture that this is due to the substitution of propensity weights q_i with the new constants q'_i in the SVM PropDCG CCP iterations. This serves as implicit variance control in the IPS estimator similar to clipping [11] by preventing propensity weights from getting too big. Since variance dominates estimation error at small amounts of data and bias dominates at large amounts, our conjecture is consistent with the observed trend.

5.3 How much presentation bias can be tolerated?

We now vary the severity of the presentation bias via η – higher values leading to click propensities more skewed to the top positions – to understand its impact on the learning algorithm. Figure 4 shows the impact on DCG performance for both methods. We report performance for two training set sizes that differ by a factor of 5 (noise $\epsilon_- = 0$). We see that SVM PropDCG is at least as robust to

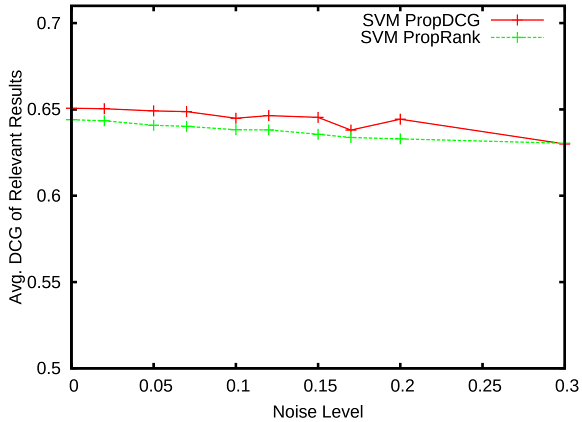

Figure 5: Test set Avg DCG performance for SVM PropDCG and SVM PropRank as the noise level increases in terms of ϵ ($n = 170K, \eta = 1$).

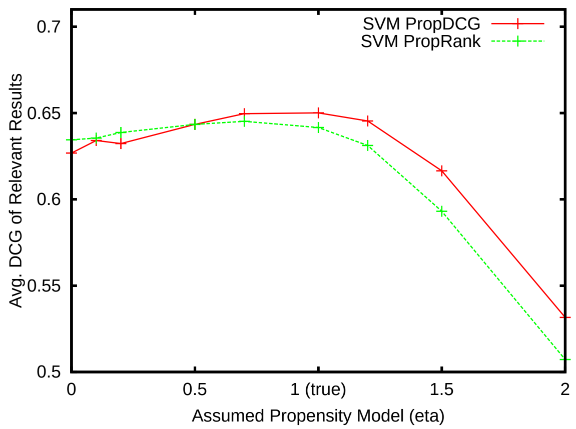

Figure 6: Test set Avg DCG performance for SVM PropDCG and SVM PropRank as propensities are misspecified (true $eta = 1, n = 170K, \epsilon_- = 0.1$).

the severity of bias as SVM PropRank. In fact, SVM PropRank's performance degrades more at high bias than that of SVM PropDCG, further supporting the conjecture that the DCG weighting in SVM PropDCG provides improved variance control which is especially beneficial when propensity weights are large. Furthermore, as also noted for SVM PropRank in [11], increasing the amount of training data by a factor of 5 improves performance of both methods due to variance reduction, which is an advantage that unbiased learning methods have over those that optimize a biased objective.

5.4 How robust is SVM PropDCG to noise?

Figure 5 shows the impact of noise on DCG performance, as noise levels in terms of ϵ_- increase from 0 to 0.3. The latter results in click data where 59.8% of all clicks are on irrelevant documents. As expected, performance degrades for both methods as noise increases. However, there is no evidence that SVM PropDCG is less robust to noise than the baseline SVM PropRank.

5.5 How robust is SVM PropDCG to misspecified propensities?

So far all experiments have had access to the true propensities that generated the synthetic click data. However, in real-world settings propensities need to be estimated and are necessarily subject to modeling assumptions. So, we evaluate the robustness of the learning algorithm to propensity misspecification.

Figure 6 shows the performance of SVM PropDCG and SVM PropRank when the training data is generated with $\eta = 1$, but the propensities used in learning are misspecified according to the η on the x-axis. The results show that SVM PropDCG is at least as robust to misspecified propensities as SVM PropRank. Both methods degrade considerably in the high bias regime when small propensities are underestimated – this is often tackled by clipping [11]. It is worth noting that SVM PropDCG performs better than SVM PropRank when misspecification leads to propensities that are underestimated, further strengthening the implicit variance control conjecture for SVM PropDCG discussed above.

5.6 How well does the CCP converge?

Next, we consider the computational efficiency of employing the CCP optimization procedure for training SVM PropDCG. Recall that the SVM PropDCG objective is an upper bound on the regularized (negative) DCG IPS estimate. It is optimized via CCP which repeatedly solves convex subproblems using the SVM PropRank solver until the objective value converges.

In Figure 7, optimization progress vs number of iterations as indicated by the change in objective value as well as the training DCG SNIPS estimate [23] is shown for $17K$ training clicks and the full range of regularization parameter C used in validation. The figure shows that the objective value usually converges in 3-5 iterations, a phenomenon observed in our experiments for other amounts of training data as well. In fact, the convergence tends to take slightly fewer iterations for larger amounts of data in terms. The figure also shows that progress in objective is well-tracked with progress in the training DCG estimate, which suggests that the objective is a suitable upper bound for DCG optimization.

It is worth noting that restarting the optimizer across multiple CCP iterations can be substantially less time consuming than the initial solution that SVM PropRank computes. Since only the coefficients of the Quadratic Program change, the data does not need to be reloaded and the optimizer can be warm-started for quicker convergence in subsequent CCP iterations.

5.7 Does performance improve further using a two-layer neural net?

We have seen that SVM PropDCG optimizes DCG better than SVM PropRank, and that it is a robust method across a wide range of biases and noise levels. Now we explore if performance can be improved further by introducing non-linearity via neural networks. Since the point of this paper is not a specific deep architecture but a novel training objective, we used a simple two-layer neural network with 200 hidden units and sigmoid activation. We expect that specialized deep architectures will further improve performance.

Figure 8 shows that Deep PropDCG achieves improved DCG compared to the linear SVM PropDCG given enough training data.

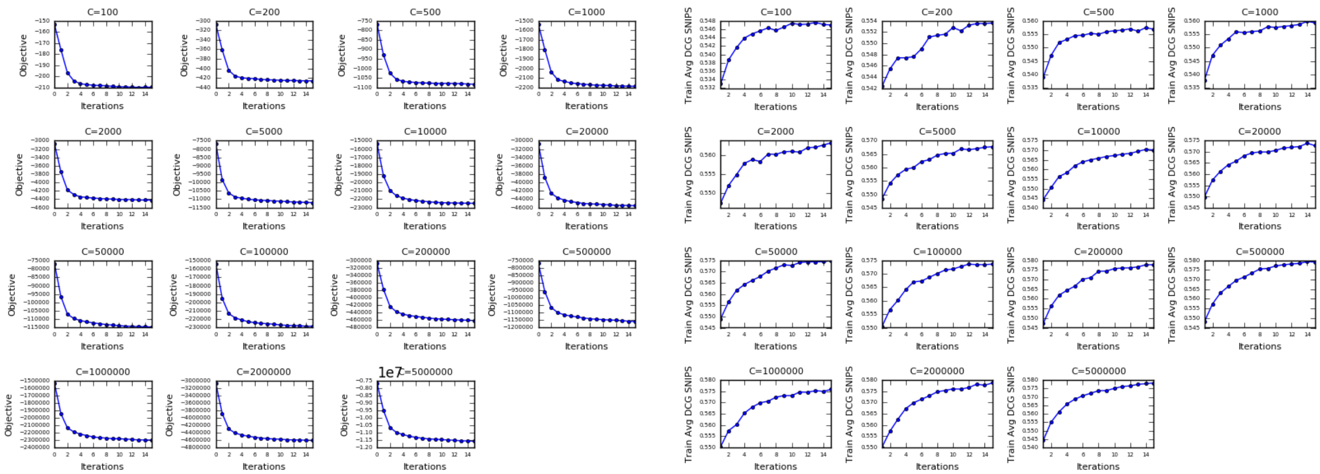

Figure 7: Optimization progress with respect to the number of CCP iterations. The objective value is shown in the left plots, and the training set DCG estimate on the right plots. Each plot corresponds to a particular value of regularization constant C ($n = 17K$, $\eta = 1$, $\epsilon_{-} = 0.1$).

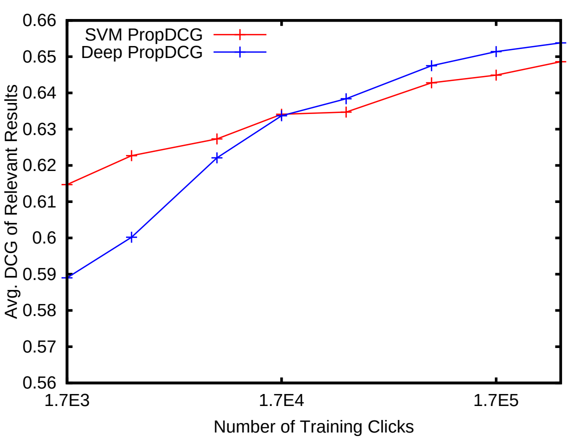

Figure 8: Test set Avg DCG performance for SVM PropDCG and Deep PropDCG ($\eta = 1$, $\epsilon_{-} = 0.1$)

For small amounts of training data, the linear model performs better, which is to be expected given the greater robustness to overfitting of linear models.

We also expect improved performance from tuning the hyperparameters of Deep PropDCG. In fact, we only used default parameters for Deep PropDCG, while we optimized the hyperparameters of SVM PropDCG on the validation set. In particular, Adam was used for stochastic gradient descent with weight decay regularizer at 10^{-6}, minibatch size of 1000 documents and 750 epochs. The learning rate began at 10^{-6} for the first 300 epochs, dropping by one order of magnitude in the next 200 epochs and another order of magnitude in the remaining epochs. We did not try any other hyperparameter settings and these settings were held fixed across varying amounts of training data.

6 CONCLUSION

In this paper, we proposed a counterfactual learning-to-rank framework that is broad enough to cover a broad class of additive IR metrics as well as non-linear deep network models. Based on the generalized framework, we developed the SVM PropDCG and Deep PropDCG methods that optimize DCG via the Convex-Concave Procedure (CCP) and stochastic gradient descent respectively. We found empirically that SVM PropDCG performs better than SVM PropRank in terms of DCG, that it is robust to a substantial amount of presentation bias, noise and propensity misspecification, and that it can be optimized efficiently. DCG was improved further by using a simple two-layer neural network in Deep PropDCG.

There are many directions for future work. First, it is open for which other ranking metrics it is possible to develop efficient and effective methods using the generalized counterfactual framework. Second, the general counterfactual learning approach may also provide unbiased learning objectives for other settings beyond ranking, like full-page optimization and browsing-based retrieval tasks. Finally, it is an open question whether non-differentiable (e.g. tree-based) ranking models can be trained in the counterfactual framework as well.

REFERENCES

[1] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *Proceedings of the 25th International Conference on World Wide Web*. 531–541.
[2] 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 (ICML '05)*. ACM, New York, NY, USA, 89–96.
[3] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*. 193–200.
[4] Olivier Chapelle and Mingrui Wu. 2010. Gradient Descent Optimization of Smoothed Information Retrieval Metrics. *Inf. Retr.* 13, 3 (June 2010), 216–235. https://doi.org/10.1007/s10791-009-9110-3
[5] Olivier Chapelle and Ya Zhang. 2009. A dynamic bayesian network click model for web search ranking. In *International Conference on World Wide Web (WWW)*.

ACM, 1–10.
[6] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search.* Morgan & Claypool Publishers.
[7] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An Experimental Comparison of Click Position-bias Models. In *International Conference on Web Search and Data Mining (WSDM)*. ACM, 87–94.
[8] D. G. Horvitz and D. J. Thompson. 1952. A Generalization of Sampling Without Replacement from a Finite Universe. *J. Amer. Statist. Assoc.* 47, 260 (1952), 663–685.
[9] T. Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 133–142.
[10] T. Joachims, T. Finley, and Chun-Nam Yu. 2009. Cutting-Plane Training of Structural SVMs. *Machine Learning* 77, 1 (2009), 27–59.
[11] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. ACM, New York, NY, USA, 781–789.
[12] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms. In *International Conference on Web Search and Data Mining (WSDM)*. 297–306.
[13] Thomas Lipp and Stephen Boyd. 2016. Variations and extension of the convex–concave procedure. *Optimization and Engineering* 17, 2 (2016), 263–287.
[14] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.
[15] Alistair Moffat and Justin Zobel. 2008. Rank-biased Precision for Measurement of Retrieval Effectiveness. *TOIS* 27, 1 (2008), 2:1–2:27.
[16] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli. 2011. SortNet: Learning to Rank by a Neural Preference Function. *IEEE Transactions on Neural Networks* 22, 9 (Sept 2011), 1368–1380.
[17] Paul R. Rosenbaum and Donald B. Rubin. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika* 70, 1 (1983), 41–55.
[18] T. Schnabel, A. Swaminathan, P. Frazier, and T. Joachims. 2016. Unbiased Comparative Evaluation of Ranking Functions. In *ACM International Conference on the Theory of Information Retrieval (ICTIR)*.
[19] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims. 2016. Recommendations as Treatments: Debiasing Learning and Evaluation. In *International Conference on Machine Learning (ICML)*.
[20] B. Schoelkopf and A. J. Smola. 2002. *Learning with Kernels.* The MIT Press, Cambridge, MA.
[21] A. Swaminathan and T. Joachims. 2015. Batch Learning from Logged Bandit Feedback through Counterfactual Risk Minimization. *Journal of Machine Learning Research (JMLR)* 16 (Sep 2015), 1731–1755.
[22] A. Swaminathan and T. Joachims. 2015. Counterfactual Risk Minimization: Learning from Logged Bandit Feedback. In *WWW Workshop on Offline and Online Evaluation of Web-based Services*.
[23] A. Swaminathan and T. Joachims. 2015. The Self-Normalized Estimator for Counterfactual Learning. In *Neural Information Processing Systems (NIPS)*.
[24] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. SoftRank: Optimizing Non-smooth Rank Metrics. *WSDM '08* (2008).
[25] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*. ACM.
[26] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *WSDM*.
[27] Yue Wang, Dawei Yin, Luo Jie, Pengyuan Wang, Makoto Yamada, Yi Chang, and Qiaozhu Mei. 2016. Beyond Ranking: Optimizing Whole-Page Presentation. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (WSDM '16)*. 103–112.
[28] Mingrui Wu, Yi Chang, Zhaohui Zheng, and Hongyuan Zha. 2009. Smoothing DCG for Learning to Rank: A Novel Approach Using Smoothed Hinge Functions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. ACM, New York, NY, USA, 1923–1926. https://doi.org/10.1145/1645953.1646266
[29] Yisong Yue, T. Finley, F. Radlinski, and T. Joachims. 2007. A Support Vector Method for Optimizing Average Precision. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 271–278.