Abstract

The accuracy of information retrieval systems is often measured using complex loss functions such as the average precision (AP) or the normalized discounted cumulative gain (NDCG). Given a set of positive and negative samples, the parameters of a retrieval system can be estimated by minimizing these loss functions. However, the non-differentiability and non-decomposability of these loss functions does not allow for simple gradient based optimization algorithms. This issue is generally circumvented by either optimizing a structured hinge-loss upper bound to the loss function or by using asymptotic methods like the direct-loss minimization framework. Yet, the high computational complexity of loss-augmented inference, which is necessary for both the frameworks, prohibits its use in large training data sets. To alleviate this deficiency, we present a novel quicksort flavored algorithm for a large class of non-decomposable loss functions. We provide a complete characterization of the loss functions that are amenable to our algorithm, and show that it includes both AP and NDCG based loss functions. Furthermore, we prove that no comparison based algorithm can improve upon the computational complexity of our approach asymptotically. We demonstrate the effectiveness of our approach in the context of optimizing the structured hinge loss upper bound of AP and NDCG loss for learning models for a variety of vision tasks. We show that our approach provides significantly better results than simpler decomposable loss functions, while requiring a comparable training time.

1. Introduction

Information retrieval systems require us to rank a set of samples according to their relevance to a query. The risk of the predicted ranking is measured by a user-specified loss function. Several intuitive loss functions have been proposed in the literature. These include simple decomposable losses (that is, loss functions that decompose over each training sample) such as 0-1 loss [16, 20] and the area under the ROC curve [1, 11], as well as the more complex non-decomposable losses (that is, loss functions that depend on the entire training data set) such as the average precision (AP) [5, 27] and the normalized discounted cumulative gain (NDCG) [6].

When learning a retrieval system, one can use a training objective that is agnostic to the risk, such as in the case of LambdaMART [4]. In this work, we focus on approaches that explicitly take into account the loss function used to measure the risk. Such approaches can use any one of the many machine learning frameworks such as structured support vector machines (SSVM) [24, 25], deep neural networks [23], decision forests [14], or boosting [21]. To estimate the parameters of the framework, they employ a training objective that is closely related to the empirical risk computed over a large training data set. Specifically, it is common practice to employ either a structured hinge upper bound to the loss function [6, 27], or an asymptotic alternative such as direct loss minimization [10, 22].

The feasibility of both the structured hinge loss and the direct loss minimization approach depends on the computational efficiency of the loss-augmented inference procedure. When the loss function is decomposable, the loss-augmented inference problem can be solved efficiently by independently considering each training sample. However, for non-decomposable loss functions, it presents a hard computational challenge. For example, given a training data set with \( P \) positive (relevant to the query) and \( N \) negative (not relevant to the query) samples, the best known algorithms for loss-augmented inference for AP and NDCG loss functions have a complexity of \( O(PN + N \log N) \) [6, 27]. Since the number of negative samples \( N \) can be very large in practice, this prohibits their use on large data sets.

In order to address the computational challenge of non-decomposable loss functions such as those based on AP and
NDCG, we make three contributions. First, we characterize a large class of ranking based loss functions that are amenable to a novel quicksort flavored optimization algorithm for the corresponding loss-augmented inference problem. We refer to the class of loss functions as \textit{QS-suitable}. Second, we show that the AP and the NDCG loss functions are QS-suitable, which allows us to reduce the complexity of the corresponding loss-augmented inference to \(O(N \log P)\). Third, we prove that there cannot exist a comparison based method for loss-augmented inference that can provide a better asymptotic complexity than our quicksort flavored approach. It is worth noting that our work is complementary to previous algorithms that have been proposed for the AP based loss functions [18]. Specifically, while the method of [18] cannot improve the asymptotic complexity of our loss-augmented inference algorithm, it can be used to reduce the runtime of a subroutine. For the sake of clarity, we limit our discussion to the structured hinge loss upper bound to the loss function. However, as our main contribution is to speed-up loss-augmented inference, it is equally applicable to direct loss minimization. We demonstrate the efficacy of our approach on the challenging problems of action recognition, object detection and image classification, using publicly available data sets. Rather surprisingly, we show that in case of some models, parameter learning by optimizing complex non-decomposable AP and NDCG loss functions can be carried out faster than by optimizing simple decomposable 0-1 loss. Specifically, while each loss-augmented inference call is more expensive for AP and NDCG loss functions, it can take fewer calls in practice to estimate the parameters of the corresponding model.

2. Background

We begin by providing a brief description of a general retrieval framework that employs a rank-based loss function, hereby referred to as the ranking framework. Note that this framework is the same as or generalizes the ones employed in previous works [6, 12, 18, 22, 27]. The two specific instantiations of the ranking framework that are of interest to us employ the average precision (AP) loss and the normalized discounted cumulative gain (NDCG) loss respectively. A detailed description of the two aforementioned loss functions is provided in the subsequent subsection.

2.1. The Ranking Framework

Input. The input to this framework is a set of \(n\) samples, which we denote by \(X = \{x_i, i = 1, \ldots, n\}\). For example, each sample can represent an image and a bounding box of a person present in the image. In addition, we are also provided with a query, which in our example could represent an action such as ‘jumping’. Each sample can either belong to the positive class (that is, the sample is relevant to the query) or the negative class (that is, the sample is not relevant to the query). For example, if the query represents the action ‘jumping’ then a sample is positive if the corresponding person is performing the jumping action, and negative otherwise. The set of positive and the negative samples are denoted by \(\mathcal{P}\) and \(\mathcal{N}\) respectively, which we assume are provided during training, but are not known during testing.

Output. Given a query and a set of \(n\) samples \(X\), the desired output of the framework is a ranking of the samples according to their relevance to the query. This is often represented by a ranking matrix \(R \in \{-1, 0, 1\}^{n \times n}\) such that \(R_{x,y} = 1\) if \(x\) is ranked higher than \(y\), \(-1\) if \(x\) is ranked lower than \(y\) and \(0\) if \(x\) and \(y\) are ranked the same. In other words, the matrix \(R\) is an anti-symmetric that represents the relative ranking of a pair of samples.

Given the sets \(\mathcal{P}\) and \(\mathcal{N}\) during training, we construct a ground truth ranking matrix \(R^*\), which ranks each positive sample above all the negative samples. Formally, the ground truth ranking matrix \(R^*\) is defined such that \(R^*_{x,y} = 1\) if \(x \in \mathcal{P}\) and \(y \in \mathcal{N}\), \(-1\) if \(x \in \mathcal{N}\) and \(y \in \mathcal{P}\), and \(0\) if \(x, y \in \mathcal{P}\) or \(x, y \in \mathcal{N}\). Note that the ground truth ranking matrix only defines a partial ordering on the samples since \(R^*_{x,y} = 0\) for all pairs of positive and negative samples. We will refer to rankings where no two samples are ranked equally as \textit{proper rankings}. Without loss of generality, we will treat all rankings other than the ground truth one as a proper ranking by breaking ties arbitrarily.

Discriminant Function. Given an input set of samples \(X\), the discriminant function \(F(X, R; w)\) provides a score for any candidate ranking \(R\). Here, the term \(w\) refers to the parameters of the discriminant function. We assume that the discriminant function is piecewise differentiable with respect to its parameters \(w\). One popular example of the discriminant function used throughout the ranking literature is the following:

\[
F(X, R; w) = \frac{1}{|\mathcal{P}| |\mathcal{N}|} \sum_{x \in \mathcal{P}} \sum_{y \in \mathcal{N}} R_{x,y} (\phi(x; w) - \phi(y; w)).
\]  

(1)

Here, \(\phi(x; w)\) is the score of an individual sample, which can be provided by a structured SVM or a deep neural network with parameters \(w\).

Prediction. Given a discriminant function \(F(X, R; w)\) with parameters \(w\), the ranking of an input set of samples \(X\) is predicted by maximizing the score, that is, by solving the following optimization problem:

\[
R(w) = \arg\max_{R} F(X, R; w). \tag{2}
\]

The special form of the discriminant function in equation (1) enables us to efficiently obtain the predicted ranking \(R(w)\) by sorting the samples in descending order of their
individual scores $\phi(x; w)$. We refer the reader to [12, 27] for details.

**Parameter Estimation.** We now turn towards estimating the parameters of our model given input samples $X$, together with their classification into positive and negative sets $P$ and $N$ respectively. To this end, we minimize the risk of prediction computed using a user-specified loss function $\Delta(R^*, R(w))$, where $R^*$ is the ground truth ranking that is determined by $P$ and $N$ and $R(w)$ is the predicted ranking as shown in equation (2). We estimate the parameters of our model as

$$w^* = \min_w \mathbb{E}[\Delta(R^*, R(w))] .$$

In the above equation, the expectation is taken with respect to the data distribution.

**Optimization for Parameter Estimation.** For many intuitive rank based loss functions such as AP loss and NDCG loss, owing to their non-differentiability and non-decomposability, problem (3) can be difficult to solve using simple gradient based methods. One popular approach is to modify problem (3) to instead minimize a structured hinge loss upper bound to the user-specified loss. We refer the reader to [27] for further details about this approach.

Formally, the model parameters can now be obtained by solving the following problem:

$$w^* = \min_w \mathbb{E}[J(w)]$$

$$J(w) = \max R^* \mathbb{E}[\Delta(R^*, R) + F(X, R; w) - F(X, R^*; w)]$$

The function $J(w)$ in problem (4) is continuous and piecewise differentiable, and is amenable to gradient based optimization. The semi-gradient of $J(w)$ takes the following form:

$$\nabla_w J(w) = \nabla_w F(X, R; w) - \nabla_w F(X, R^*; w) ,$$

with,

$$R = \arg\max R^* \mathbb{E}[\Delta(R^*, R) + F(X, R; w)] .$$

Borrowing terminology from the structured prediction literature [13, 27], we call $R$ the most violating ranking and problem (6) as the loss-augmented inference problem. An efficient procedure for loss-augmented inference is key to solving problem (4).

While we focus on using loss-augmented inference for estimating the semi-gradient, it can also be used as the cutting plane [13] and the conditional gradient of the dual of problem (4). In addition to this, loss-augmented inference is also required for solving problem (3) using the direct loss minimization framework [22].

---

1For a continuous function $f(x)$ defined on a domain of any generic dimension, we can define semi-gradient $\nabla_s f(x)$ to be a random picking from the set $\{\nabla f(t) : ||x - t|| < \epsilon\}$, for a sufficiently small $\epsilon$.

2Chakrabarti et al. [6] use a slightly modified definition of the discount $D(.)$. For a detailed discussion about it the reader can refer to the Appendix (Supplementary).
negative. For the ranking $\mathbf{R}$ that induces the order (7), we can compute
\[
\Delta_{\text{NDCG}}(\hat{\mathbf{R}}, \mathbf{R}) = 1 - \frac{1 + \log_2^{-1} 3 + \log_2^{-1} 5 + \log_2^{-1} 7}{1 + \log_2^{-1} 3 + \log_2^{-1} 4 + \log_2^{-1} 5} 
\approx 0.056.
\]
Both AP loss and NDCG loss are functions of the entire dataset and are not decomposable onto individual samples.

3. Quicksort Flavored Optimization

In order to estimate the parameters $\mathbf{w}$ in the ranking framework by solving problem (4), we need to compute the semi-gradient of $J(\mathbf{w})$. To this end, given the current estimate of parameters $\mathbf{w}$, as well as a set of samples $\mathbf{X}$, we are interested in obtaining the most violated ranking by solving problem (6). At first glance, the problem seems to require us to obtain a ranking matrix $\hat{\mathbf{R}}$. However, it turns out that we do not explicitly require a ranking matrix.

In more detail, our algorithm uses an intermediate representation of the ranking using the notion of interleaving ranks. Given a ranking $\mathbf{R}$ and a negative sample $x$, the interleaving rank $\text{rank}(x)$ is defined as one plus the number of positive samples preceding $x$ in $\mathbf{R}$. Note that, similar to our notation for $\text{ind}(\cdot)$, $\text{ind}^+(\cdot)$ and $\text{ind}^-(\cdot)$, we have dropped the dependency of $\text{rank}(\cdot)$ on $\mathbf{R}$ as the ranking matrix would be clear from context. The interleaving rank of all the samples does not specify the total ordering of all the samples according to $\mathbf{R}$ as it ignores the relative ranking of the positive samples among themselves, and the relative ranking of the negative samples among themselves. However, as will be seen shortly, for a large class of ranking based loss functions, interleaving ranks corresponding to the most violating ranking are sufficient to compute the semi-gradient as in equation (5).

In the rest of the section, we discuss the class of loss functions that are amenable to a quicksort flavored algorithm, which we call QS-suitable loss functions. We then describe and analyze our quicksort flavored approach for finding the interleaving rank in some detail. For brevity and simplicity of exposition, in the following sub-section, we restrict our discussion to the properties of QS-suitable loss functions that are necessary for an intuitive explanation of our algorithm. For a thorough discussion on the characterization and properties of QS-suitable loss functions, we refer the interested reader to the full version of the paper [19].

3.1. QS-Suitable Loss Functions

As discussed earlier, many popular rank-based loss functions happen to be non-decomposable. That is, they cannot be additively decomposed onto individual samples. However, it turns out that a wide class of such non-decomposable loss functions can be instead additively decomposed onto the negative samples. Formally, for some functions $\delta_j : \{1, \ldots, |P| + 1\} \to \mathbb{R}$ for $j = 1, \ldots, |N|$, for a proper ranking $\mathbf{R}$ one can write
\[
\Delta(\mathbf{R}, \mathbf{R}) = \sum_{x \in N} \delta_{\text{ind}^-(x)}(\text{rank}(x)).
\]
We will call this the negative-decomposability property.

Further, many of those rank-based loss functions do not depend on the relative order of positive or negative samples among themselves. Rather, the loss for a ranking $\mathbf{R}$, $\Delta(\mathbf{R}, \mathbf{R})$, depends only on the interleaving rank of positive and negative samples corresponding to $\mathbf{R}$. We will call this the interleaving-dependence property.

As will be evident later in the section, the above properties in a loss function allows for an efficient quicksort flavored divide and conquer algorithm to solve the loss augmented problem. We formally define the class of loss functions that allow for such a quicksort flavored algorithm as QS-suitable loss functions. The following proposition establishes the usefulness for such a characterization.

**Proposition 1** Both $\Delta_{\text{AP}}$ and $\Delta_{\text{NDCG}}$ are QS-suitable.

The proof of the above proposition is provided in Appendix (supplementary). Having established that both the AP and the NDCG loss are QS-suitable, the rest of the section will deal with a general QS-suitable loss function. A reader who is interested in employing another loss function need only check whether the required conditions are satisfied in order to use our approach.

3.2. Key Observations for QS-Suitable Loss

Before describing our algorithm in detail, we first provide some key observations which enable efficient optimization for QS-suitable loss functions. To this end, let us define an array $\{s_i^+\}_{i=1}^{|P|}$ of positive sample scores and an array $\{s_i^-\}_{i=1}^{|N|}$ of negative sample scores. Furthermore, for purely notational purposes, let $\{s^*_i\}$ be the array $\{s_i^-\}$ sorted in descending order. For $j \in \{1, \ldots, |N|\}$ we denote the index of $s_j^-$ in $\{s_i^+\}$ as $j^*$.

With the above notation, we describe some key observations regarding QS-suitable loss functions. Their proofs are for most part straightforward generalizations of results that appeared in [18, 27] in the context of the AP loss and can be found in Appendix (Supplementary). Using the interleaving-dependence property of QS-suitable loss functions and structure of the discriminant function as defined in equation (1), we can make the following observation.

**Observation 1** An optimal solution $\hat{\mathbf{R}}$ of problem (6) would have positive samples appearing in the descending order of their scores $s_i^+$ and also the negative samples appearing in descending order of their scores $s_i^-$. Now, in order to find the optimal ranking $\hat{\mathbf{R}}$, it would seem natural to sort the arrays $\{s_i^+\}$ and $\{s_i^-\}$ in descending order and then find the optimal interleaving ranks $\text{rank}(x)$
for all \( x \in \mathcal{N} \). However, we are aiming for complexity below \( O(|\mathcal{N}| \log |\mathcal{N}|) \), therefore we can not afford to sort the negative scores. On the other hand, since \( |\mathcal{P}| << |\mathcal{N}| \), we are allowed to sort the array of positive scores \( \{s_i^+\} \).

Let \( \text{opt}_i \) be the optimal interleaving rank for the negative sample with the \( i^{th} \) rank in the sorted list \( \{s_i^+\} \) and \( \text{opt} = \{\text{opt}_i | i = 1, \ldots, |\mathcal{N}|\} \) be the optimal interleaving rank vector. A certain subtle monotonicity property Q5-suitable loss functions (see Supplementary) and the structure of the discriminant function given in (1) gives us the opportunity to compute the interleaving rank for each negative sample independently. However, we actually need not do this computation for all the \( |\mathcal{N}| \) negative samples. This is because, since the interleaving rank for any negative sample can only belong to \([1, |\mathcal{P}| + 1]\) and \( |\mathcal{P}| << |\mathcal{N}| \), many of the negative samples would have the same interleaving rank. This fact can be leveraged to improve the efficiency of the algorithm for finding \( \text{opt} \) by making use of the following observation.

**Observation 2** If \( i < j \), then \( \text{opt}_i \leq \text{opt}_j \).

Knowing that \( \text{opt}_i = \text{opt}_j \) for some \( i < j \), we can conclude that \( \text{opt}_i = \text{opt}_k = \text{opt}_j \) for each \( i < k < j \). This provides a cheap way to compute some parts of the vector \( \text{opt} \) if an appropriate sequence is followed for computing the interleaving ranks. Even without access to the fully sorted set \( \{s_i^+\} \), we can still find \( s_j^+ \), the \( j \)-th highest element in \( \{s_i^+\} \), for a fixed \( j \), in \( O(|\mathcal{N}|) \) time. This would lead to an \( O(|\mathcal{P}| |\mathcal{N}|) \) algorithm but we may at each step modify \( \{s_i^+\} \) slowly introducing the correct order. This will make the future searches for \( s_j^+ \) more efficient.

### 3.3. Divide and Conquer

Algorithm 1 describes the main steps of our approach. Briefly, we begin by detecting \( s_{|\mathcal{N}|/2}^+ \) that is the median score among the negative samples. We use this to compute \( \text{opt}_{|\mathcal{N}|/2} \). Given \( \text{opt}_{|\mathcal{N}|/2} \), we know that for all \( j < |\mathcal{N}|/2 \), \( \text{opt}_j \in [1, \text{opt}_{|\mathcal{N}|/2}] \) and for all \( j > |\mathcal{N}|/2 \), \( \text{opt}_j \in [\text{opt}_{|\mathcal{N}|/2}, |\mathcal{P}| + 1] \). This observation allows us to employ a divide-and-conquer recursive approach.

In more detail, we use two classical linear time array manipulating procedures \textsc{median} and \textsc{select}. The first one outputs the index of the median element. The second one takes as its input an index of a particular element \( x \). It rearranges the array such that \( x \) separates higher-ranked elements from lower-ranked elements (in some total order). For example, if array \( s^- \) contains six scores \([a \ b \ 4.5 \ 6 \ 1 \ c]\) then \text{median}(3, 5) would return 3 (the index of score 4.5), while calling \text{select}(3, 5) would rearrange the array to \([a \ b \ 4.5 \ 6 \ c]\) and return 4 (the new index of 4.5). The \text{select} procedure is a subroutine of the classical \textsc{quicksort} algorithm.

Using the two aforementioned procedures in conjunction with the divide-and-conquer strategy allows us to compute the entire interleaving rank vector \( \text{opt} \) and this in turn allows us to compute the semi-gradient \( \nabla_w J(w) \), as in equation (5), efficiently.

Figure 1 provides an illustrative example of our divide-and-conquer strategy. Here, \( |\mathcal{N}| = 11 \) and \( |\mathcal{P}| = 2 \). We assume that the optimal interleaving rank vector \( \text{opt} \) is \([1, 2, 2, 2, 2, 2, 2, 2, 3, 3] \). Let us now go through the procedure in which Algorithm 1 computes this optimal interleaving rank vector. Before starting the recursive procedure, we only sort the positive samples according to their scores and do not sort the negative samples. To start with, we call \text{opt}Ranks(1, 11, 1, 3). We find the negative sample with the median score (6th highest in this case) and compute its optimal interleaving rank \( \text{opt}_6 \) to be 2. In the next step of the recursion, we make the following calls: \text{opt}Ranks(1, 5, 1, 2) and \text{opt}Ranks(7, 11, 2, 3). These calls compute \( \text{opt}_3 \) and \( \text{opt}_9 \) to be 2. In the next set of recursion calls however, the calls \text{opt}Ranks(4, 5, 2, 2) and \text{opt}Ranks(7, 8, 2, 2), get terminated in step 4 of Algorithm 1 and \( \text{opt}_j \) for \( j = 4, 5, 7, 8 \) are assigned without any additional computation. We then continue this procedure recursively for progressively smaller intervals as described in Algorithm 1. Leveraging the fact stated in observation 2, our algorithm has to explicitly compute the interleaving rank for only 6 (shown in square brackets) out of the 11 negative samples. In a typical real data set, which is skewed more in favor of the negative samples, the expected number of negative samples for which the interleaving rank has to be explicitly computed is far less than \( |\mathcal{N}| \). In contrast, the algorithm proposed by Yue et al. in

---

**Algorithm 1: Recursive procedure for finding all interleaving ranks.**

**Description:** The function finds optimal interleaving rank for all \( i \in [\ell^-, r^-] \) given that

(i) array \( s^- \) is partially sorted, namely \( \max(s^-[\ell^--1], \ldots, \ell^-) \leq \min(s^-[\ell^-, \ldots, r^-]) \) and \( \max(s^-[\ell^-, \ldots, r^-]) \leq \min(s^-[r^-+1, \ldots, |\mathcal{N}|]) \);

(ii) optimal interleaving ranks for \( i \in [\ell^-, r^-] \) lie in the interval \([\ell^+, r^+]\).

1. function \text{OptRanks}(\text{int} \ell^-, \text{int} r^-, \text{int} \ell^+, \text{int} r^+)
2.     if \( \ell^+ = r^+ \) then
3.         set \( \text{opt}_i = \ell^+ \) for each \( i \in [\ell^-, r^-] \) and return
4.     end
5.     \( m = \text{Median}(\ell^-, r^-) \) \( \triangleright \) gives the index of the median score in a subarray of \( s^- \)
6.     \( m = \text{Select}(m, \ell^-, r^-) \) \( \triangleright \) splits the subarray by \( s = s^-[m] \), returns the new index of \( s \)
7.     Find \( \text{opt}_m \) by trying all options in \([\ell^+, \ell^+]\)
8.     if \( \ell^- < m \) then \text{OptRanks}(\ell^-, m-1, \ell^+, \text{opt}_m)
9.     if \( m < r^- \) then \text{OptRanks}(m+1, r^-, \ell^+, \text{opt}_m, + r^+)

---


of 4846 images, which include 10 different action classes. The data set is divided into two parts: 3347 ‘trainval’ person bounding boxes and 3363 ‘test’ person bounding boxes. We use the ‘trainval’ bounding boxes for training since their ground-truth action classes are known. We evaluate the accuracy of the different models on the ‘test’ bounding boxes using the PASCAL evaluation server.

Model. We use structured SVM models as discriminant functions and use the standard poselet [17] activation features to define the sample feature for each person bounding box. The feature vector consists of 2400 action poselet activations and 4 object detection scores. We refer the reader to [17] for details regarding the feature vector.

Methods. We show the effectiveness of our method in optimizing both AP loss and NDCG loss to learn the model parameters. Specifically, we report the computational time for the loss-augmented inference evaluations. For AP loss, we compare our method (referred to as AP, QS) with the loss-augmented inference procedure described in [27] (referred to as AP). For NDCG loss, we compare our method (referred to as NDCG, QS) with the loss-augmented inference procedure described in [6] (referred to as NDCG). We also report results for loss-augmented inference evaluations when using the simple decomposable 0-1 loss function (referred to as 0-1). The hyperparameters involved are fixed using 5-fold cross-validation on the ‘trainval’ set.

Results. When we minimize AP loss on the training set to learn the model parameters, we get a mean AP of 51.196 on the test set. In comparison, minimizing 0-1 loss to learn model parameters leads to a mean AP value of 47.934 on the test set. Similarly, minimizing NDCG loss for parameter learning gives a superior mean NDCG value of 85.521 on the test set, compared to that of 84.3823 when using 0-1 loss. The AP and NDCG values obtained on the test set for individual action classes can be found in the supplementary material. This clearly demonstrates the usefulness of directly using rank based loss functions like AP loss and NDCG loss for learning model parameters, instead of using simple decomposable loss functions like 0-1 loss as surrogates.

The time required for the loss augmented inference evalu-

<table>
<thead>
<tr>
<th>0-1</th>
<th>AP</th>
<th>AP, QS</th>
<th>NDCG</th>
<th>NDCG, QS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0094</td>
<td>0.7154</td>
<td>0.0825</td>
<td>6.8019</td>
<td>0.0473</td>
</tr>
</tbody>
</table>

Table 1. Total computation time (in seconds) when using the different methods, for multiple calls to loss augmented inference during model training. The reported time is averaged over the training for all the action classes.

<table>
<thead>
<tr>
<th>0-1</th>
<th>AP</th>
<th>AP, QS</th>
<th>NDCG</th>
<th>NDCG, QS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.48±0.03</td>
<td>16.29±0.18</td>
<td>1.48±0.59</td>
<td>71.07±1.57</td>
<td>0.35±0.11</td>
</tr>
</tbody>
</table>

Table 2. Mean computation time (in milliseconds) when using the different methods, for single call to loss augmented inference. The reported time is averaged over all training iterations and over all the action classes.
Table 1. Comparison of 0-1 losses, AP and AP_QS.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Computation time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>AP_QS</td>
</tr>
<tr>
<td>NDCG</td>
<td>NDCG_QS</td>
</tr>
<tr>
<td>0-1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. Total computation time for multiple calls to loss augmented inference during model training, when the number of total, negative and positive samples are varied. Here, 0-1, AP and AP_QS correspond to loss augmented inference procedures for 0-1 loss, for AP loss using [27] and for AP loss using our method respectively. It can be seen that our method scales really well with respect to sample set sizes and takes computational time that is comparable to what is required for simpler 0-1 decomposable loss.

Figure 3. Total computation time for multiple calls to loss augmented inference during model training, when the number of total, negative and positive samples are varied. Here, 0-1, NDCG and NDCG_QS correspond to loss augmented inference procedures for 0-1 loss, for NDCG loss using [6] and for NDCG loss using our method respectively. As can be seen, our approach scales elegantly with respect to sample set sizes and is comparable to the simpler 0-1 decomposable loss in terms of computation time.

4.2. Object Detection

Data set. We use the PASCAL VOC 2007 [8] object detection data set, which consists of a total of 9963 images. The data set is divided into a 'trainval' set of 5011 images and a 'test' set of 4952 images. All the images are labeled to indicate the presence or absence of the instances of 20 different object categories. In addition, we are also provided with tight bounding boxes around the object instances, which we ignore during training and testing. Instead, we treat the location of the objects as a latent variable. In order to reduce the latent variable space, we use the selective-search algorithm [26] in its fast mode, which generates an average of 2000 candidate windows per image. This results in a training set size of approximately 10 million bounding boxes.

Model. For each candidate window, we use a feature representation that is extracted from a trained Convolutional Neural Network (CNN). Specifically, we pass the image as input to the CNN and use the activation vector of the penultimate layer of the CNN as the feature vector. Inspired by the R-CNN pipeline of Girshick et al. [9], we use the CNN that is trained on the ImageNet data set [7], by rescaling each candidate window to a fixed size of $224 \times 224$. The length of the resulting feature vector is 4096. However, in contrast to [9], we do not assume ground-truth bounding boxes to be available for training images. We instead optimize AP loss in a weakly supervised framework to learn the parameters of
the SVM based object detectors for the 20 object categories.

Methods. We use our approach to learn the parameters of latent AP-SVMS [2] for each object category. In our experiments, we fix the hyperparameters using 5-fold cross-validation. During testing, we evaluate each candidate window generated by selective search and use non-maxima suppression to prune highly overlapping detections.

Results. For this task of weakly supervised object detection, using AP loss for learning model parameters leads to a mean test AP of 36.616 which is significantly better than the 29.4995 obtained using 0-1 loss. The AP values obtained on the test set by the detectors for each object class can be found in the supplementary material. These results establish the usefulness of optimizing AP loss for learning the object detectors. On the other hand, optimizing AP loss for this task places high computational demands due to the size of the data set (5011 ‘trainval’ images) as well as the latent space (2000 candidate windows per image) amounting to around 10 million bounding boxes. We show that using our method for loss-augmented inference (LAI) leads to significant saving in computational time. During training, the total time taken for LAI, averaged over all the 20 classes, was 0.5214 sec for our method which is an order of magnitude better than the 7.623 sec taken by the algorithm proposed in [27]. Thus, using our efficient quicksort flavored algorithm can be critical when optimizing non-decomposable loss functions like AP loss for large scale data sets.

4.3. Image Classification

Data set. We use the CIFAR-10 data set [15], which consists of a total of 60,000 images of size 32 × 32 pixels. Each image belongs to one of 10 specified classes. The data set is divided into a ‘trainval’ set of 50,000 images and a ‘test’ set of 10,000 images. From the 50,000 ‘trainval’ images, we use 45,000 for training and 5,000 for validation. For our experiments, all the images are centered and normalized.

Model. We use a deep neural network as our classification model. Specifically, we use a piecewise linear convolutional neural network (PL-CNN) as proposed in [3]. We follow the same framework as [3] for experiments on the CIFAR-10 data set and use a PL-CNN architecture comprising 6 convolutional layers and an SVM last layer. For all our experiments, we use a network that is pre-trained using softmax and cross-entropy loss.

Methods. We learn the weights of the PL-CNN by optimizing AP loss and NDCG loss for the training data set. For comparison, we also report results for parameter learning using the simple decomposable 0-1 loss. We use the layerwise optimization algorithm called LW-SVM, proposed in [3], for optimizing the different loss functions with respect to the network weights. Following the training regime used in [3], we warm start the optimization with a few epochs of Adadelta [28] before running the layer wise optimization. The LW-SVM algorithm involves solving a structured SVM problem for one layer at a time. This requires tens of thousands of calls to loss augmented inference and having a efficient procedure is therefore critical for scalability. We compare our method for loss-augmented inference with the methods described in [27] and [6], for AP loss and NDCG loss respectively.

Results. We get a better mean AP of 85.28 on the test set when we directly optimize AP loss for learning network weights compared to that of 84.22 for 0-1 loss. Similarly, directly optimizing NDCG loss leads to a better mean NDCG of 96.14 on the test set compared to 95.31 for 0-1 loss. This establishes the usefulness of optimizing non-decomposable loss functions like the AP loss and NDCG loss. The LW-SVM algorithm involves very high number of calls to the loss augmented inference procedure. In light of this, the efficient method for loss augmented inference proposed in this paper leads to significant reduction in total training time. When optimizing the AP loss, using our method leads to a total training time of 1.589 hrs compared to that of 1.974 hrs for the algorithm proposed in [27]. Similarly, when optimizing NDCG loss, our method leads to a total training time of 1.632 hrs, which is significantly better than the 2.217 hrs taken for training when using the method proposed in [6]. This indicates that using our method helps the layerwise training procedure scale much better.

5. Discussion

We provided a characterization of ranking based loss functions that are amenable to a quicksort based optimization algorithm for the loss augmented inference problem. We proved that our algorithm provides a better computational complexity than the state of the art methods for AP and NDCG loss functions and also established that the complexity of our algorithm cannot be improved upon asymptotically by any comparison based method. We empirically demonstrated the efficacy of our approach on challenging real world vision problems. In future, we would like to explore extending our approach to other ranking based non-decomposable loss functions like those based on the F-measure or the mean reciprocal rank.

Acknowledgement

This work is partially funded by the EPSRC grants EP/P020658/1 and TU/B/000048 and a CEFIPRA grant. MR and VK were supported by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 616160.
Appendix

The supplementary material is organized as follows. In Section 1, we give a full definition of QS-suitable loss functions and in Section 2 we justify the correctness of Algorithm 1. Section 3 contains proofs of QS-suitability of AP and NDCG losses and Section 4 establishes worst-case complexity of Algorithm 1 as well as a matching lower-bound on the complexity of solving Problem (6). Several remaining proofs are delegated to Section 5 and we use Section 6 for certain clarifications regarding previous use of NDCG in the literature. Finally, we report some additional experimental results in Section 7.

1. Complete characterization of QS-Suitable Loss Functions

A proper loss function $\Delta = \Delta(R^*, R)$ is called QS-suitable if it meets the following three conditions.

(C1) Negative decomposability with interleaving dependence. There are functions $\delta_j : \{1, \ldots, |P| + 1\} \rightarrow R$ for $j = 1, \ldots, |N|$ such that for a proper ranking $R$ one can write

$$\Delta(R^*, R) = \sum_{x \in N} \delta_{\text{ind}^-(x)}(\text{rank}(x)).$$

(C2) $j$-monotonicity of discrete derivative. For every $1 \leq j < |N|$ and $1 \leq i \leq |P|$ we have

$$\delta_{j+1}(i + 1) - \delta_{j+1}(i) \geq \delta_j(i + 1) - \delta_j(i).$$

(C3) Fast evaluation of discrete derivative. For any $j \in \{1, \ldots, |N|\}$ and $i \in \{1, \ldots, |P|\}$, can the value $\delta_j(i + 1) - \delta_j(i)$ be computed in constant time.

From (C1), we can see that the loss function depends only on the interleaving ranks of the negative samples. More accurately, it depends on the vector $r = (r_1, \ldots, r_{|N|})$ where $r_i$ is the interleaving rank of the $i$-th most relevant negative sample (i.e. with the $i$-th highest score).

Another way to interpret this type of dependence is by looking at the $\pm$-pattern of a ranking which can be obtained as follows. Given a proper ranking $R$ (in the form of a permutation of samples), it is the pattern obtained by replacing each positive sample with a “+” symbol and each negative sample with a “-” symbol. It is easy to see that the $\pm$-pattern uniquely determines the vector $r$ and vice versa and thus (C1) also implies dependence on the $\pm$-pattern.

2. Justification of Algorithm 1

First key point is that the entire objective function (6) inherits properties (C1) and (C2).

Observation 3 The following holds:

(a) There are functions $f_j : \{1, \ldots, |P| + 1\} \rightarrow R$ for $j = 1, \ldots, |N|$ such that the objective function in (6) can be written as

$$\sum_{j=1}^{|N|} f_j(r_j),$$

where $r_j$ is the interleaving rank of the negative sample $x$ with $\text{ind}^-(x) = j$.

(b) The functions $f_j$ inherit property (C2). More precisely, for every $1 \leq j < |N|$ and $1 \leq i \leq |P|$ we have

$$f_{j+1}(i + 1) - f_{j+1}(i) \geq f_j(i + 1) - f_j(i).$$

(c) We can compute $\arg\max_{1 \leq i \leq r} f_j(i)$ in $O(r - l)$ time if we are provided access to the sorted array $\{s^+_i\}$ and to the score of the negative sample $x$ with $\text{ind}^-(x) = j$.

As a result, solving Problem (6) reduces to computing the optimal interleaving ranks (or the optimal vector $r$ from the remark above) $^3$.

The next vital point is that these interleaving ranks can be optimized independently. This is however not obvious. One certainly can maximize each $f_j$ but the resulting vector $r$ may not induce any ranking – its entries may not be monotone.

But as a matter of fact, this does not happen and Observation 2 from the main text gives the precise guarantee. This “correctness of greedy maximization” hinges upon condition (C2) as will also be demonstrated with a counterexample given later in Section 6.

All in all, it suffices to compute the vector $\text{opt}$ in which $\text{opt}_j = \max \arg\max_{1 \leq i \leq r} f_j$ (the maximum ensures that ties are broken consistently) as is done in the main text of the paper.

3. Properties of $\Delta_{\text{AP}}$ and $\Delta_{\text{NDCG}}$

In this place, let us prove the aforementioned properties of $\Delta_{\text{AP}}$ and $\Delta_{\text{NDCG}}$.

Proposition 3 $\Delta_{\text{NDCG}}$ is QS-suitable.

Proof. As for (C1), let us first verify that the functions $\delta_j$ can be set as

$$\delta_j(i) = \frac{1}{\mathcal{O}}(D(i + j - 1) - D(|P| + j)),$$

$^3$Note that the value of the objective can be computed efficiently given a vector $r$ – for example by constructing any ranking $R$ which respects $r$. 
where $C = \sum_{i=1}^{[P]} D(i)$. Indeed, one can check that
\[
\Delta(R^*, R') = 1 - \frac{\sum_{x \in P} D(\text{ind}(x))}{\sum_{i=1}^{[P]} D(i)}
\]
\[
= 1 - \frac{1}{C} \sum_{i=1}^{[P]} D(i) - \frac{\sum_{x \in P} D(\text{ind}(x)) + \text{rank}(x) - 1}{\sum_{x \in N} \delta_{\text{ind}^{-1}(x)}(\text{rank}(x))}
\]
as desired. As for (C2) and (C3), let us realize that
\[
\delta_j(i+1) - \delta_j(i) = \frac{1}{C} (D(i+j) - D(i+j-1)).
\]
Then (C3) becomes trivial and checking (C2) reduces to
\[
D(i+j+1) + D(i+j) = 2D(i+j)
\]
which follows from convexity of the function $D$. □

**Proposition 4** $\Delta_{AP}$ is QS-suitable.

**Proof.** Regarding (C1), the functions $\delta_j$ were already identified in [27] as
\[
\delta_j(i) = \frac{1}{[P]} \sum_{k=i}^{[P]} \left( \frac{j}{j+k} - \frac{j-1}{j+k-1} \right)
\]
so after writing
\[
\delta_j(i+1) - \delta_j(i) = \frac{j-1}{j+i-1} - \frac{j}{j+i}
\]
we again have (C3) for free and (C2) reduces to
\[
2g_l(j) \geq g_l(j-1) + g_l(j+1),
\]
where $g_l(x) = \frac{1}{x+1}$, and the conclusion follows from concavity of $g_l(x)$ for $x > 0$. □

**4. Computational Complexity**

Now is the time to establish the computational complexity of Algorithm 1 as well as the afore-mentioned matching lower bound, efficiency.

**Theorem 5** If $\Delta$ is QS-suitable, then the Problem (6) can be solved in time $O(|N| \log |P| + |P| \log |P| + |P| \log |N|)$, which in the most common case $|N| > |P|$ reduces to $O(|N| \log |P|)$.

Outside running Algorithm 1, the entire computation also consists of preprocessing (sorting positive samples by their scores) and post processing (computing the output from vector opt). These subroutines have only one non-linear complexity term – $O(|P| \log |P|)$ coming from the sorting. Therefore, it remains to establish the complexity of Algorithm 1 as $O(|N| \log |P| + |P| \log |N|)$.

To this end, let us denote $n = r^- - \ell^- + 1$ and $p = r^+ - \ell^+ + 1$, and set $T_{\text{neg}}(n, p), T_{\text{pos}}(n, p)$ as the total time spent traversing the arrays of negative and positive sample scores, respectively, including recursive calls. The negative score array is traversed in the MEDIAN and SELECT procedures and the positive scores are traversed when searching for opt. The latter has by complexity $O(p)$, due to Observation 3(c), whose assumption are always satisfied during the run of the algorithm.

**Proposition 6** The runtimes $T_{\text{neg}}(n, p)$ and $T_{\text{pos}}(n, p)$ satisfy the following recursive inequalities
\[
T_{\text{neg}}(n, p) \leq Cn + T_{\text{neg}}(n/2, p_1) + T_{\text{neg}}(n/2, p_2)
\]
for some $p_1 + p_2 = p + 1$,
\[
T_{\text{pos}}(n, p) \leq Cp + T_{\text{pos}}(n/2, p_1) + T_{\text{pos}}(n/2, p_2)
\]
for some $p_1 + p_2 = p + 1$,
\[
T_{\text{neg}}(n, 1) \leq Cn, \quad T_{\text{neg}}(1, p) = 0,
\]
\[
T_{\text{pos}}(n, 1) = 0, \quad T_{\text{pos}}(1, p) \leq Cp
\]
for a suitable constant $C$. These inequalities imply $T_{\text{neg}}(n, p) \leq C'n \log(1 + p)$ and $T_{\text{pos}}(n, p) \leq C'(p - 1) \log(1 + n)$ for another constant $C'$. Thus the running time of Algorithm 1, where $p = |P| + 1$, $n = |N|$, is $O(|N| \log |P| + |P| \log |N|)$.

**Proof.** The recursive inequalities follow from inspection of Algorithm 1. As for the “aggregated” inequalities, we proceed in both cases by induction. For the first inequality the base step is trivial for high enough constant $C'$ and for the inductive step we may write
\[
T_{\text{neg}}(n, p) \leq Cn + T_{\text{neg}}(n/2, p_1) + T_{\text{neg}}(n/2, p_2)
\]
\[
\leq Cn + \frac{1}{2} C'n \log(1 + p_1) + \frac{1}{2} C'n \log(1 + p_2)
\]
\[
\leq C'n \left( \frac{C}{C'} + \log \sqrt{(1 + p_1)(1 + p_2)} \right)
\]
\[
\leq C'n \log(p_1 + p_2) = C'n \log(1 + p)
\]
where in the last inequality we used that
\[
1 + (1 + p_1)(1 + p_2) \leq (p_1 + p_2)^2
\]
for integers $p_1, p_2$ with $p_1 + p_2 = p + 1 \geq 3$. That makes the last inequality true for sufficiently high $C'$ (not depending on $n$ and $p$).

The proof of the second inequality is an easier variation on the previous technique. □
4.1. Lower Bound on Complexity

In order to prove the matching lower bound (among comparison-based algorithms), we intend to use the classical information theoretical argument: There are many possible outputs and from each comparison we receive one bit of information, therefore we need “many” comparison to shatter all output options.

Proposition 7 Let $\Delta$ be a loss function. Then any comparison-based algorithm for Problem (6) requires $\Omega(|\mathcal{N}| \log |\mathcal{P}|)$ operations.

Proof. Since the negative samples are unsorted on the input and the scores are arbitrary, every possible mapping from $\{1, \ldots, |\mathcal{N}|\}$ to $\{1, \ldots, |\mathcal{P}| + 1\}$ may induce the (unique) optimal assignment of interleaving ranks. There are $(|\mathcal{P}| + 1)^{|\mathcal{N}|}$ possibilities to be distinguished and each comparison has only two possible outcomes. Therefore we need $\log_2 \left((|\mathcal{P}| + 1)^{|\mathcal{N}|}\right) \in \Omega(|\mathcal{N}| \log |\mathcal{P}|)$ operations. □

5. Remaining proofs

Throughout the text we omitted several proofs, mostly because they are straightforward generalizations of what already appeared in [27] and [6]. For the sake of completeness, we present them here.

Proof of Observation 1 (of main text): Let $\mathbf{R}$ be any optimal solution. We check that $F(\mathbf{X}, \mathbf{R}; \mathbf{w})$ increases if we swap two samples $x, y \in \mathcal{P}$ in $\mathbf{R}$ with $\text{ind}(x) < \text{ind}(y)$ and $\phi(x; \mathbf{w}) < \phi(y; \mathbf{w})$ (it boils down to $ac + bd > ad + bc$ for $a > b \geq 0$ and $c > d \geq 0$). Since similar argument applies for negative samples, we can conclude that $\mathbf{R}$ already has both negative and positive samples sorted decreasingly. Otherwise, one could perform swaps in $\mathbf{R}$ that would increase the value of the objective, a contradiction with the optimality of $\mathbf{R}$. □

Proof of Observation 2 (of main text): Recall that $\text{opt}_j$ is the highest rank with maximal value of the corresponding $f_j$. It suffices to prove that for $i_{j+1} = \max \text{argmax } f_{j+1}$ and $i_j = \max \text{argmax } f_j$, we have $i_{j+1} \geq i_j$. Since by Observation 3 functions $f_j$ inherit property (C2), we can compare the discrete derivatives of $f_j$ and $f_{j+1}$, all left to do is to formalize the discrete analogue of what seems intuitive for continuous functions.

Assume $i_{j+1} < i_j$. Then since $f_{j+1}(i_j) - f_{j+1}(i_{j+1}) = \sum_{i = i_{j+1}}^{i_j - 1} f_{j+1}(i + 1) - f_{j+1}(i) \geq \sum_{i = i_{j+1}}^{i_j - 1} f_j(i + 1) - f_j(i) = f_j(i_j) - f_j(i_{j+1}) \geq 0$, we obtain that $i_j \in \text{argmax } f_{j+1}$ and as $i_j > i_{j+1} = \max \text{argmax } f_{j+1}$ and we reached the expected contradiction. □

Lemma 8 The objective function $F(\mathbf{X}, \mathbf{R}; \mathbf{w})$ decomposes into contributions of negative and positive samples as follows:

$$F(\mathbf{X}, \mathbf{R}; \mathbf{w}) = \frac{1}{|\mathcal{P}| |\mathcal{N}|} \sum_{x \in \mathcal{P}} \sum_{y \in \mathcal{N}} R_{x,y}(\phi(x; \mathbf{w}) - \phi(y; \mathbf{w})) = \sum_{x \in \mathcal{P}} c(x)\phi(x; \mathbf{w}) + \sum_{y \in \mathcal{N}} c(y)\phi(y; \mathbf{w}),$$

where

$$c(x) = \frac{|\mathcal{N}| + 2 - 2\text{rank}(x)}{|\mathcal{P}| |\mathcal{N}|},$$
$$c(y) = \frac{|\mathcal{P}| + 2 - \text{rank}(y)}{|\mathcal{P}| |\mathcal{N}|}.$$ 

In particular, assuming already that $\{s_i^+\}$ is sorted, and that $\mathbf{R}$ is induced by a vector of interleaving ranks $\mathbf{r}$, one has

$$F(\mathbf{X}, \mathbf{R}; \mathbf{w}) = \sum_{i=1}^{|\mathcal{P}|} c_i^+ s_i^+ + \sum_{j=1}^{|\mathcal{N}|} c_j^- s_j^-,$$

where

$$c_i^+ = \frac{|\mathcal{N}| + 2 - 2r_i^+}{|\mathcal{P}| |\mathcal{N}|},$$
$$c_j^- = \frac{|\mathcal{P}| + 2 - 2r_j^-}{|\mathcal{P}| |\mathcal{N}|}.$$ 

Here $r_i^+$ stands for the interleaving rank of the $i$-th positive sample, which can be computed as $r_i^+ = 1 + |\{j : r_j \leq i\}|$.

Proof. This is straightforward to verify with a short computation. □

Proof of Observation 3: We slightly modify the decomposition from Lemma 8 in order to incorporate the array $\{s_i^+\}$:

$$F(\mathbf{X}, \mathbf{R}; \mathbf{w}) = \sum_{y \in \mathcal{N}} c(y)\phi(y; \mathbf{w}) + \sum_{x \in \mathcal{P}} R_{x,y}(\phi(x; \mathbf{w})) = \frac{1}{|\mathcal{N}| |\mathcal{P}|} \sum_{j=1}^{|\mathcal{N}|} \left(|\mathcal{P}| + 2 - \sum_{i=1}^{r_j-1} s_i^+ + \sum_{i=1}^{|\mathcal{P}|} s_i^- - \sum_{i=1}^{|\mathcal{N}|} s_i^+\right).$$
This, in combination with (C1), defines the functions $f_j$ for $j = 1, \ldots, |\mathcal{N}|$. As for the condition (C2), we have

$$f_j(i + 1) - f_j(i) = \frac{2(s_i^+ - s_i^-)}{|\mathcal{N}| |\mathcal{P}|} + \delta_j(i + 1) - \delta_j(i),$$

where, let us be reminded, $\{s_i^+\}$ is the sorted array of scores of negative samples. After writing analogous equality for $j + 1$ and using that (C2) holds for functions $\delta_j$, we can check that the desired inequality

$$f_{j+1}(i + 1) - f_{j+1}(i) \geq f_j(i + 1) - f_j(i)$$

follows from $s_{j+1}^+ \leq s_j^+$.

Note that for computing the $\arg\max f_j(i)$ it is sufficient to compute all discrete derivatives (i.e. all the differences $f_j(i + 1) - f_j(i)$); the actual values of $f_j$ are in fact not needed. For $\delta_j$ we know that one such evaluation is constant time and for $f_j$ this is also the case since we assumed to have access to $s_j^+$.

\[ \square \]

6. NDCG and Discount Functions

Chakrabarti et al. [6] use a slightly modified definition of the discount $D(\cdot)$ as

$$D(i) = \begin{cases} 1 & 1 \leq i \leq 2 \\ 1/\log_2(i) & i > 2 \end{cases}.$$ 

For the resulting NDCG loss, a greedy algorithm is proposed for solving the loss augmented inference problem. This algorithm achieves the runtime of $O(|\mathcal{N}| |\mathcal{P}| + |\mathcal{N}| \log |\mathcal{N}|)$.

The authors also suggest to use a cut-off $k$ in the definition of discount $D(i)$, setting $D(i) = 0$ for $i \geq k$. With this simplification they achieved a reduced complexity of $O((|\mathcal{N}| + |\mathcal{P}|) \log(|\mathcal{P}| + |\mathcal{N}|) + k^2)$.

However, with the above definition of a discount, it is possible to obtain a counter-case where their proof of correctness of the greedy algorithm is not valid (specifically, there exists a counter-example for Fact 3.4 of [6]). For the greedy algorithm to be correct, it turns out that the convexity of $D(i)$ is essential.

Remark 1 Observation 2 is not true for $\Delta_{NDCG}$ with function $D(i)$ taken as

$$D(i) = \begin{cases} 1 & 1 \leq i \leq 2 \\ 1/\log_2(i) & i > 2 \end{cases}.$$ 

Proof. Consider negative samples $x_1$ and $x_2$ and a positive sample $x_3$ with scores $s_1 = 3\varepsilon$, $s_2 = \varepsilon$, $s_3 = 5\varepsilon$, where $\varepsilon > 0$ is small.

Note that the NDCG loss of a ranking $R$ reduces to $\Delta_{NDCG}(R^*, R) = 1 - D(\text{ind}(x_3))$ where we used the fact that $D(1) = 1$.

The decomposition $\Delta_{\text{NDCG}}(R^*, R) = \delta_1(r_1) + \delta_2(r_2)$ holds if we set

$$\delta_1(1) = \delta_2(1) = 0,$$

$$\delta_1(2) = D(2) - D(3),$$

$$\delta_2(2) = D(1) - D(2) = 0$$

and (possibly by looking at the proof of Observation 3) we also find values of $f_1$ and $f_2$ as

$$f_1(1) = \frac{1}{2} (s_1 - s_3) + \delta_1(1) = -\varepsilon < \varepsilon$$

$$f_2(1) = \frac{1}{2} (s_2 - s_3) + \delta_2(1) = -2\varepsilon + D(2) - D(3) > 2\varepsilon$$

$$f_2(2) = \frac{1}{2} (s_3 - s_2) + \delta_2(2) = f_2(2).$$

Hence $opt_1 = 2 > 1 = opt_2$, a contradiction.

\[ \square \]

7. Additional Experimental Results

For the action classification experiments on the PASCAL VOC 2011 data set, we report the performance of models trained by optimizing 0-1 loss as well as AP loss in Table 3.

For the object detection experiments, we report the detection AP in Table 5 for all the 20 object categories obtained by models trained using 0-1 loss as well as AP loss. For all object categories other than 'bottle', AP loss based training does better than that with 0-1 loss. For 15 of the 20 object categories, we get statistically significant improvement with AP loss trained models compared to those trained using 0-1 loss (using paired t-test with p-value less than
0.05). While optimizing AP loss for learning gives an overall improvement of 7.12% compared to when using 0-1 loss, for 5 classes it gives an improvement of more than 10%. The bottom 2 classes with the least improvement obtained by AP loss based training, ‘chair’ and ‘bottle’ seem to be difficult object categories to detect, with detectors registering very low detection APs. In conjunction with the overall superior performance of AP loss for learning model parameters, the efficient method proposed by this paper makes a good case for optimizing AP loss rather than 0-1 loss for tasks like object detection.

<table>
<thead>
<tr>
<th>Object class</th>
<th>0-1 loss</th>
<th>AP loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jumping</td>
<td>86.409</td>
<td>48.18</td>
</tr>
<tr>
<td>Phoning</td>
<td>73.134</td>
<td>61.45</td>
</tr>
<tr>
<td>Playing instrument</td>
<td>81.533</td>
<td>36.73</td>
</tr>
<tr>
<td>Reading</td>
<td>74.528</td>
<td>19.66</td>
</tr>
<tr>
<td>Riding bike</td>
<td>94.928</td>
<td>1.01</td>
</tr>
<tr>
<td>Running</td>
<td>93.766</td>
<td>49.51</td>
</tr>
<tr>
<td>Taking photo</td>
<td>74.058</td>
<td>66.78</td>
</tr>
<tr>
<td>Using computer</td>
<td>79.518</td>
<td>40.77</td>
</tr>
<tr>
<td>Walking</td>
<td>89.789</td>
<td>3.23</td>
</tr>
<tr>
<td>Riding horse</td>
<td>96.160</td>
<td>38.52</td>
</tr>
</tbody>
</table>

Table 4. Performance of classification models trained by optimizing 0-1 loss and NDCG loss, in terms of NDCG on the test set for the different action classes of PASCAL VOC 2011 action dataset. We conduct 5-fold cross-validation and report the mean NDCG over the five validation sets.

<table>
<thead>
<tr>
<th>Object category</th>
<th>0-1 loss</th>
<th>AP loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aeroplane</td>
<td>46.60</td>
<td>48.18</td>
</tr>
<tr>
<td>Bicycle</td>
<td>48.53</td>
<td>61.45</td>
</tr>
<tr>
<td>Bird</td>
<td>33.31</td>
<td>36.73</td>
</tr>
<tr>
<td>Boat</td>
<td>15.23</td>
<td>19.66</td>
</tr>
<tr>
<td>Bottle</td>
<td>6.10</td>
<td>1.01</td>
</tr>
<tr>
<td>Bus</td>
<td>37.01</td>
<td>49.51</td>
</tr>
<tr>
<td>Car</td>
<td>61.28</td>
<td>66.78</td>
</tr>
<tr>
<td>Cat</td>
<td>38.12</td>
<td>40.77</td>
</tr>
<tr>
<td>Chair</td>
<td>2.71</td>
<td>3.23</td>
</tr>
<tr>
<td>Cow</td>
<td>21.06</td>
<td>38.52</td>
</tr>
<tr>
<td>Dining-table</td>
<td>14.20</td>
<td>39.53</td>
</tr>
<tr>
<td>Dog</td>
<td>33.55</td>
<td>36.25</td>
</tr>
<tr>
<td>Horse</td>
<td>46.14</td>
<td>53.86</td>
</tr>
<tr>
<td>Motorbike</td>
<td>29.97</td>
<td>34.81</td>
</tr>
<tr>
<td>Person</td>
<td>29.58</td>
<td>30.41</td>
</tr>
<tr>
<td>Potted-plant</td>
<td>21.27</td>
<td>23.03</td>
</tr>
<tr>
<td>Sheep</td>
<td>11.65</td>
<td>32.20</td>
</tr>
<tr>
<td>Sofa</td>
<td>36.66</td>
<td>42.03</td>
</tr>
<tr>
<td>Train</td>
<td>29.71</td>
<td>37.10</td>
</tr>
<tr>
<td>TV-monitor</td>
<td>27.31</td>
<td>37.26</td>
</tr>
</tbody>
</table>

Table 5. Performance of detection models trained by optimizing 0-1 loss and AP loss, in terms of AP on the test set for the different object categories of PASCAL VOC 2007 test set.

References


[27] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, 2007. 1, 2, 3, 4, 6, 7, 8, 10, 11