On the Local Optimality of LambdaRank

Pinar Donmez
School of Computer Science, Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

Krysta M. Svore
Microsoft Research
One Microsoft Way
Redmond, WA 98052

Christopher J. C. Burges
Microsoft Research
One Microsoft Way
Redmond, WA 98052

Microsoft Research Technical Report MSR-TR-2008-179

November 25, 2008

Abstract

A machine learning approach to rank learning trains a model to optimize a target evaluation measure with repect to training data. Currently, existing information retrieval measures are impossible to optimize directly except for models with a trivial number of parameters. The IR community thus faces a major challenge: how to optimize IR measures of interest directly. In this paper, we present a solution. Specifically, we show that LambdaRank [1], which smoothly approximates the *gradient* of the target measure, can be adapted to work with three popular IR target evaluation measures using the same underlying gradient construction. It is likely, therefore, that this construction is extendable to other evaluation measures. We empirically show that LambdaRank finds a locally optimal solution for NDCG, MAP and MRR with a 99% confidence rate. We also show that the amount of effective training data varies with IR measure and that with a sufficiently large training set size, matching the training optimization measure to the target evaluation measure yields the best accuracy.

1 Introduction

Learning to rank is an increasingly popular area of research. Ranking is a mapping from a set of items to an ordered list; the ranking of Web search results is a common example. This task consists of a set of queries and a set of retrieved documents for each query. The document-query pairs are labeled according to a scale from not relevant to highly relevant, and the ranking systems use the training data to compute a model that outputs a rank order based on a real-valued scoring function $f(x)$. It is more important to predict the correct ordering rather than the value of $f(x)$. In Web search ranking, the cost function is typically defined with respect to a sorted order of documents at the query level, and averaged over a large number of queries.

The ranking problem generally employs a cost function (target evaluation measure) that is not necessarily the one used to train the system. Typical target measures used in IR (see [8] for a detailed list) depend only on the sorted list and the relevance levels of the listed items. These measures are generally either flat everywhere or non-differentiable with respect to the model parameters; hence they are difficult to optimize directly. One way to address this issue is to find a close smooth approximation to the target measure, and optimize it via gradient descent. However, this is quite challenging due to the sort component of the target ranking measures. LambdaRank [1] tackles the problem by defining a smooth approximation to the *gradient* of the target cost instead of searching for a smooth approximation to the target cost itself. The basic idea of LambdaRank is to specify rules determining how the rank order of documents should change. These rules are incorporated into a $\lambda-$gradient that defines the gradient of an implicit cost function only at the points of interest [1]. LambdaRank was originally proposed for NDCG (Normalized Discounted Cumulative Gain), but the method is general and works with any target cost function. Recently, LambdaRank was shown to be locally optimal for NDCG [12].

In this paper, we define $\lambda-$gradients for three widely used IR measures, namely NDCG, MAP and MRR, using the same underlying construction as used for the NDCG λ-gradient in [1]. This construction is likely extendable to other IR measures as well. We empirically show, with a confidence bound, the local optimality of LambdaRank on these measures by monitoring the change in training accuracy as we vary the learned weights of the net. We change the weights by projecting in a random direction on a unit ball and moving the weights in that direction. If the accuracy decreases as the original net weights change, it means the learned weights are at a local optimum. By checking the accuracy decreases for several hundred random directions, we show, with 99% confidence, that the learned net weights are at a local optimum, using a Monte-Carlo test with one-sided error. We also show that the gradient vanishes at each learned weight by fixing all but one weight's

value and varying that weight's value while checking for a decrease in accuracy on the training set. If the highest accuracy is achieved at the learned weight value, then the gradient has vanished. We find LambdaRank to be locally optimal and that the gradient vanishes for all weights. Our work is not only the first to show empirical optimality of a learning algorithm, but also the first to show optimality across several IR measures. In addition, it shows IR practitioners can now directly optimize for the IR measure they care about, and the model need not be limited to only a few parameters. We also show that with large enough amounts of training data, the best test accuracy is achieved when matching the training optimization measure to the target evaluation measure.

The paper is organized as follows: we review related rank learning literature in Section 2 and the IR measures under consideration in Section 3. We describe in detail the $\lambda-$gradients in Section 4. Local optimality testing is described in Section 5. Our datasets are described in Section 6 and experimental results are presented in Section 7. We conclude in Section 8 with final remarks and future work.

2 Related Work

The ranking task has become increasingly popular among researchers in the past few years. Some ranking algorithms approach the problem as structured output prediction such as the large margin methods of [9, 10]. The learned structures are mapped to the reals, and then the best structure is chosen to give the highest real-valued score among all possible outputs. Another line of work casts the ranking problem as ordinal regression, that is, learning the mapping of an input vector to a member of an ordered set of numerical ranks [6]. Like many other ranking algorithms, their cost functions depend on pairs of examples. Crammer and Singer [5] proposed a similar solution where the ranker is a perceptron whose output is a weight vector $\boldsymbol{w}$. Cao et al. [4] proposed a listwise approach to rank learning where a cross-entropy loss is defined between two parametrized probability distributions of permutations. Qin et al. [7] proposed a method called RankCosine, which depends on a listwise loss function that takes the cosine similarity between the score vectors of the predicted result and the ground truth.

In addition, there are methods that claim to directly optimize the evaluation measures, such as SVMMAP [13] and AdaRank [11]. SVMMAP incorporates MAP into the listwise optimization constraints. In reality, the number of constraints is exponential in the number of rankings. SVMMAP tackles this problem by performing optimization only on a working set of constraints which is extended with the most violated constraint at each step. The resulting algorithm works in

polynomial time. AdaRank, on the other hand, performs a boosting-type optimization where the IR measure is embedded into the loss function used in updating the distribution of the data. The authors claim a theoretical guarantee that the training error defined in terms of the IR measure will reduce constantly with some mild assumptions.

Another approach is to train on pairs of documents per query. RankNet [3] is a neural net based ranking algorithm that optimizes a cross-entropy cost function using gradient descent. It is trained on pairs of documents per query, where documents in a pair have different labels. The RankNet cost consists of a sigmoid followed by a pair-based cross-entropy cost. RankNet modifies the standard backprop algorithm by trying to minimize the value of the cost function by adjusting each weight in the net according to the gradient of the cost with respect to that weight. The training time of RankNet scales quadratically with the average number of pairs per query, and linearly with the number of queries. Thus, speeding up RankNet training becomes crucial especially for large training sets. LambdaRank [1] provides a significant training speed-up as well as a framework for optimizing a cost function while avoiding the difficulties of working with non-differentiable IR measures. In addition, LambdaRank has been shown to empirically optimize NDCG [12]. We empirically show not only that it optimizes NDCG, but also MAP and MRR.

3 IR Measures

IR measures are typically defined with respect to a permutation of documents for a given query. The relevance labels can be binary or multilevel. For binary measures, we assume labels $\{0, 1\}$ (1 for relevant, and 0 for non-relevant). Binary measures include Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Winner Takes All (WTA) (see [8] for a more complete list). In this paper, we focus on three of the most commonly used IR metrics: MAP, MRR, and NDCG.

Average Precision (AP) computes for each relevant document the precision at its position in the ranked list; these precisions are then averaged over all relevant documents:

$$\text{AP} = \frac{\sum_{r=1}^{L} l(r)\text{P@}r}{R} \tag{1}$$

where r is the rank position, L is the truncation level, R is the number of relevant documents, $l(r)$ is the binary relevance label of the document at rank position r, and P@r is the precision up to rank position r, i.e. $\text{P@}r = \frac{\sum_{i=1}^{r} l(i)}{r}$. Mean Average Precision is the average of the average precisions over all N queries, $\frac{1}{N}\sum_{i=1}^{N} \text{AP}_i$, where AP_i is the average precision for query i.

Reciprocal Rank (RR) for a given query is the reciprocal of the rank position of the highest ranking relevant document for the query. MRR is just the average of the reciprocal ranks over queries:

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i} \tag{2}$$

where N is the number of queries and r_i is the highest position of a relevant document for query i.

Unlike binary measures such as MAP and MRR, NDCG recognizes multi-relevance levels. NDCG for a given query is formulated as follows:

$$\text{NDCG@}k = \frac{1}{Z}\sum_{r=1}^{L}\frac{2^{l(r)}-1}{\log(1+r)} \tag{3}$$

where $l(r) \in \{0,\ldots,4\}$ is the relevance label of the document at rank position r. Z is chosen such that the perfect ranking would result in NDCG@$L = 1$, and L is the truncation level at which NDCG is computed. Mean NDCG@k is $\frac{1}{N}\sum_{i=1}^{N}$ NDCG@k_i, where NDCG@k_i is the NDCG@k for query i. NDCG is particularly suited for Web search applications since it accounts for multi-relevance levels and the truncation level can be set to model user behavior. In our results, we report mean NDCG@10.

4 LambdaRank

In most machine learning tasks, a target cost is used to assess the accuracy of the system at test time, and an optimization cost, generally a smooth approximation to the target cost, is used to train the system. Ideally, the optimization cost matches the target cost, but typical IR target costs (e.g. MAP, MRR, NDCG, etc.) are either flat or non-differentiable everywhere. Hence, direct optimization of the target cost is quite challenging. LambdaRank [1] solves this problem by defining the gradients of a given cost function only at the points of interest. The gradients are defined by specifying rules about how swapping two documents, after sorting them by score for a given query, changes the cost. This general method can work with any target cost function, but was originally formulated for NDCG due to its suitability for Web search applications. In this section, we define $\lambda-$gradients for three different IR measures.

4.1 $\lambda-$Gradient for NDCG

A LambdaRank gradient, λ_j, is defined to be a smooth approximation to the gradient of a target cost with respect to the score of the document at rank position j. $\lambda-$gradients have a physical interpretation; documents are represented by point masses and $\lambda-$gradients are forces on those point masses [1]. On two documents in a pair, the $\lambda-$gradients are equal and opposite, where a positive $\lambda-$gradient indicates a push toward the top of the list, and a negative $\lambda-$gradient indicates a push toward the bottom of the list. With a choice of suitable $\lambda-$gradient, the gradient of any target cost can be smoothly approximated for a given document.

The authors of [1] tried several alternatives for $\lambda-$gradients and chose the best according to accuracy on validation data (for a detailed list, the reader is referred to [2]). The best $\lambda-$gradient found in [1] is a combination of the derivative of the RankNet cost [3] scaled by the NDCG gain from swapping two documents with differing labels (for a given query). The RankNet cost is a pairwise cross-entropy cost applied to the logistic of the difference of the model scores. If document i, with score s_i, is to be ranked higher than document j, with score s_j, then the RankNet cost can be written as follows:

$$C_{ij} \equiv C(o_{ij}) = s_j - s_i + \log(1 + e^{o_{ij}}) \tag{4}$$

where $o_{ij} \equiv s_i - s_j$ is the score difference of a pair of documents in a query. The derivative of the RankNet cost according to score difference is

$$\delta C_{ij}/\delta o_{ij} = \delta C_{ij}/\delta s_i = -1/(1 + e^{o_{ij}}) \tag{5}$$

The $\lambda-$gradient can now be written as follows:

$$\lambda_{ij} \equiv S_{ij} \left| \Delta\text{NDCG} \frac{\delta C_{ij}}{\delta o_{ij}} \right| \tag{6}$$

$$= S_{ij} \left| N(2^{l(i)} - 2^{l(j)}) \left(\frac{1}{\log(1 + r_i)} - \frac{1}{\log(1 + r_j)} \right) \left(\frac{1}{1 + e^{o_{ij}}} \right) \right|$$

where N is the reciprocal max DCG for the query, l_i and l_j are the relevance labels and r_i and r_j are the rank positions of documents i and j, respectively. The sign of λ_{ij}, $S_{ij} \in \{-1, 1\}$, is plus one if $l_i > l_j$ (document i is more relevant than document j), indicating document i must move up the ranked list to reduce the cost, and minus one if $l_i < l_j$. Note that the sign only depends on the labels of documents i and j and not on their rank positions (see Eqn 5). The $\lambda-$gradient for a single document is computed by marginalizing over the pairwise $\lambda-$gradients; i.e.

$$\lambda_i = \sum_{j \in P} \lambda_{ij} \tag{7}$$

where the sum is over all pairs in a query which contain document j.

4.2 $\lambda-$Gradient for MAP

The LambdaRank algorithm is designed to work with any target cost function, as long as the $\lambda-$gradient can be defined. We design a $\lambda-$gradient for MAP based on the same principles as the one designed for NDCG, with the exception that ΔNDCG is substituted with ΔAP.

The $\lambda-$gradient for MAP uses the RankNet cost, scaled by the AP (Average Precision) gain found by swapping two documents i and j at rank positions r_i and r_j. Assume documents i and j are misranked by the current net, i.e. $r_i > r_j$ but $l(i) > l(j)$[1] then

$$\lambda_{ij} = S_{ij}\left|\frac{1}{R}\left(\sum_{k=r_j}^{r_i} l(k)\text{P@}k - \sum_{k=r_j}^{r_i} l'(k)\text{P}'@k\right)\left(\frac{1}{1+e^{o_{ij}}}\right)\right| \tag{8}$$

where $l(k) = 1$ if the document at rank position k is relevant, and 0 otherwise; P@k is the precision at rank k; R is the number of relevant documents for that query. $l'(k)$ is the relevance value after we swap the documents at positions r_i and r_j. In fact, $l'(k) = l(k)$ for all $k \in \{r_j + 1, ..., r_i - 1\}$, $l'(r_i) = l(r_j)$, and $l'(r_j) = l(r_i)$. P$'$@k is the precision at the rank positions between r_j and r_i after the swap. We can rewrite the above formula as:

$$\lambda_{ij} = S_{ij}\left|\frac{1}{R}\left[\left(\frac{n+1}{r_j} - \frac{m}{r_i}\right) + \sum_{k=r_j+1}^{r_i-1}\frac{l(k)}{k}\right]\left(\frac{1}{1+e^{o_{ij}}}\right)\right| \tag{9}$$

where n and m ($n \leq m$) are the number of relevant documents at the top r_j and the top r_i positions, respectively. The sign of λ_{ij} determines the direction of the move; i.e. relevant documents get the upward move and non-relevant documents move downward. Finally, the $\lambda-$gradient for a single document i becomes the sum of the pairwise gradients, $\lambda_i = \sum_{j \in P} \lambda_{ij}$.

4.3 $\lambda-$Gradient for MRR

The $\lambda-$gradient for MRR follows that of NDCG and MAP. It uses the RankNet cost scaled by the RR (Reciprocal Rank) gain for that query, found by swapping documents i and j at the corresponding rank positions r_i and r_j, for any $\{i, j\}$.

[1] Throughout the paper, we assume higher rank means lower rank index.

Assume document i is relevant, and document j is non-relevant,

$$\lambda_{ij} = S_{ij}\left|\Delta RR(r_i, r_j)\left(\frac{1}{1+e^{o_{ij}}}\right)\right| \tag{10}$$

ΔRR(r_i, r_j) calculates the difference in the reciprocal rank of the top relevant document as a result of the swap:

$$\Delta\text{RR}(r_i, r_j) = \begin{cases} \frac{1}{r_j} - \frac{1}{r} & \text{if } r_j < r \leq r_i \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where r is the rank of the top relevant document in the ordered list. Clearly, there is no RR gain (or loss) unless the rank of the top relevant document shifts after the swap.

5 Monte Carlo Testing of Local Optimality

In order to verify local optimality, we must show that the training accuracy of LambdaRank decreases as the weights' learned values are changed (either increased or decreased in value). We project iid random directions $\vec{r}_1, \vec{r}_2, ..., \vec{r}_k$ on a unit ball by first sampling each dimension of a vector from a Gaussian distribution[2] with 0 mean and unit variance and then projecting the vector onto the unit sphere. We modify the net weights in each direction as follows: let $\vec{w}$ be the vector of net weights, and $f_M(\vec{w})$ be the accuracy of the net with weights w with respect to a given evaluation measure M. We use a Monte-Carlo test with one-sided error; we compare, for all $i \in \{1, 2, ..., k\}$, $f_M(\vec{w})$ to $f_M(\vec{w}+\eta\vec{r}_i)$ for small $\eta > 0$ and check if $f_M(\vec{w}) \geq f_M(\vec{w} + \eta\vec{r}_i)$. If there exists at least one direction $\vec{r}_i$ for which this condition does not hold ($\vec{r}_i$ has higher accuracy), then $\vec{w}$ is not a local optimum. If the accuracy decreases for all directions $\vec{r}_i$, then the probability of missing an increasing direction is upper bounded by $(1-q)^k$, where q is a lower bound on the number of random directions k. We can further upper bound this probability by $\exp(-qk)$. We require 99% confidence that the probability of finding an increasing direction is less than 1%:

$$\exp(-0.01k) = 0.01$$

Solving for k yields $k = 460$ random directions. Thus, if we can show that varying the net weights in 460 random directions all yield worse accuracy than the learned set of weights, then the learned set of weights represents a local optimum with 99% confidence. We choose 10 different step sizes $\eta \in \{0.1, 0.2, ..., 1\}$ to analyze

[2] Any spherically symmetric distribution can be used, not just a Gaussian distribution.

Table 1: MAP scores with standard error (SE) of different $\lambda-$gradients on the validation set.

$\lambda-$gradient	MAP $\pm$ SE
RankNetWeightPairs	0.462$\pm$0.0048
LocalGradient	0.435$\pm$0.0048
LocalCost	0.427$\pm$0.0049
SpringSmooth	0.424$\pm$0.0048
DiscreteBounded	0.401$\pm$0.0049

the change in accuracy as we move away from the net weights $\vec{w}$. We compare the change in mean accuracy averaged over all queries for all 460 directions. In Section 7, we report the results for each IR measure.

6 Datasets

We conducted experiments on three different datasets. Two of them are real Web datasets from a commercial search engine. The other is an artificial dataset [3] created to remove any variance caused by the quality of features and/or relevance labels. The artificial data, first introduced in [3], was generated from random cubic polynomials. It has 300 features, 50 URLs per query, and a random 10K/5K/10K train/valid/test split for queries. We refer to it as the Artificial Data. The first Web search dataset has 26.1 URLs per query, 420 features, and 10K/5K/10K query train/valid/test splits. We call this dataset the 10K Web Data. The second Web dataset contains 30K/5K/10K query train/valid/test splits, 100 URLs per query, and 420 features. We refer to this dataset as the 30K Web Data.

All the document-query pairs are assigned integer labels between 0 (the least relevant) and 4 (the most relevant). For the binary measures, MAP and MRR, we transform the multilevel relevance levels to binary by converting all labels between 2 and 4 (inclusive) to relevant (1), and all the rest to non-relevant (0).

7 Experiments

We empirically show the local optimality of LambdaRank on three datasets for three IR measures. We also investigate if matching the training measure to the target evaluation measure yields the best test accuracy, and if so, how much training data is required. First, we select the best of 13 $\lambda-$gradient constructions [2] by training each construction for each measure on a 5K query Web set and choosing the one with the best validation accuracy on a 5K query validation set. In Table 1,

Table 2: MRR scores with standard error (SE) of different $\lambda-$gradients on the validation set.

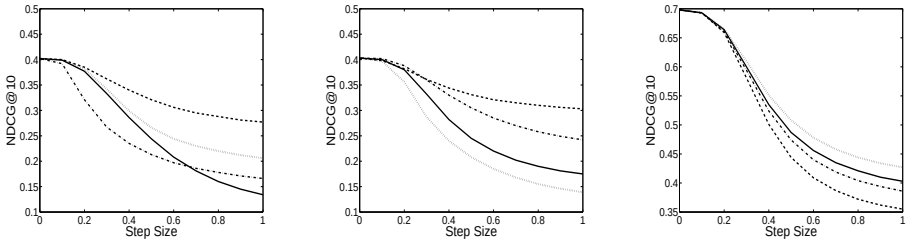
$\lambda-$gradient	MRR $\pm$ SE
RankNetWeightPairs	0.524$\pm$0.0059
LocalCost	0.515$\pm$0.0060
LocalGradient	0.512$\pm$0.0059
SpringSmooth	0.498$\pm$0.0058
DiscreteBounded	0.471$\pm$0.0059

we report the validation accuracy of a selective subset of $\lambda-$gradients for MAP. In Table 2, we compare the accuracy of $\lambda-$gradients for MRR. 'RankNetWeightPairs' is the construction explained in detail throughout Section 4. 'SpringSmooth' is a smoothed version of 'RankNetWeightPairs' where the gain obtained by swapping a pair is lower-bounded by 1. 'LocalGradient' estimates the gradient by the change in accuracy with respect to the difference in scores between two adjacent documents in an ordered list. A margin is added to handle very small score differences. 'LocalCost' uses a cost based on a document's neighbors to compute an estimate of the local gradient. 'DiscreteBounded' computes the change in accuracy when a document is moved to its ideal position in the ranked order, and the $\lambda-$gradient is upper-bounded by 1. We adopt 'RankNetWeightPairs' as the construction for NDCG, MAP, and MRR since it outperformed the other constructions, with statistical significance, on the validation set for all three measures. It is likely this construction can be extended to other IR measures as well.

7.1 Empirical Optimality of LambdaRank

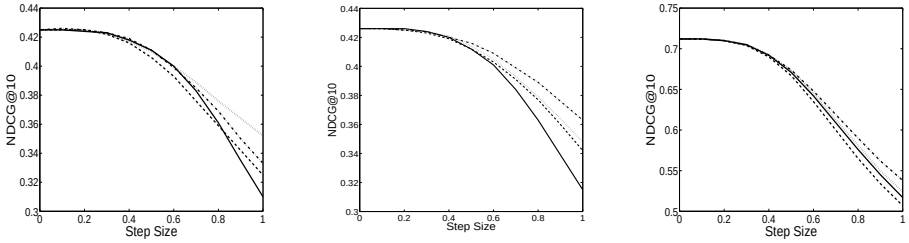
We trained LambdaRank on each training set and each evaluation measure, which we denote by *LambdaRankNDCG, LambdaRankMAP,* and *LambdaRankMRR*. For each algorithm, we varied the learning rate between 10^{-7} to 10^{-3}, and picked the rate that gave the best validation accuracy. Each algorithm was run for 700 epochs. If the training accuracy decreased at a given epoch, the learning rate was reduced by a factor of 0.8 with 30% probability. The output of each algorithm is a set of learned model weights. We report results in this paper based on both single-layer and two-layer nets with 10 hidden nodes.

Figure 1 shows the change in mean NDCG@10 on the three training sets when varying the weights of the single-layer net in a given random direction. Note from this point on we use NDCG and mean NDCG@10 interchangeably. In all figures, for readability, we graph only 4 of the 460 random directions tested. When the step size $\eta = 0$, the accuracy corresponds to the training accuracy of the original



a) 10K Web Data b) 30K Web Data c) Artificial Data

Figure 1: Shifting single-layer weights for mean NDCG@10. x-axis is the step size, η.



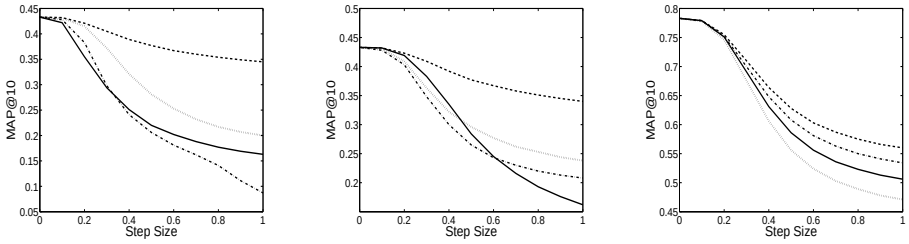
a) 10K Web Data b) 30K Web Data c) Artificial Data

Figure 2: Shifting 2-layer weights for mean NDCG@10. x-axis is the step size.

(learned) net. On all three training sets, mean NDCG@10 decreases as we change the values of the learned weights. The training accuracy score is higher on the Artificial dataset, as expected, since it is a much less noisy dataset. Figure 2 shows the training accuracy on all three training sets when varying the 2-layer weights, where we vary the hidden and first layer weights together. In all cases, NDCG curves are smooth functions of the weights and it is apparent the learned weights result in the best accuracy.

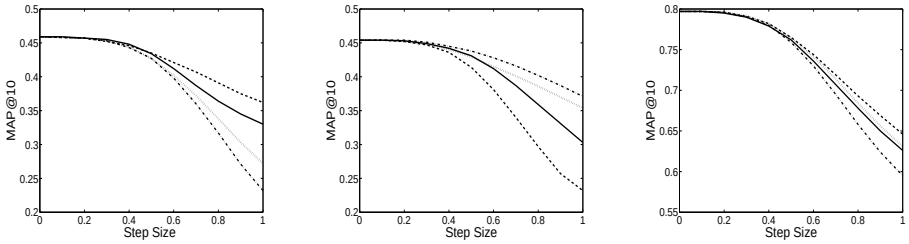
We also compare the change in accuracy with respect to MAP and MRR. Figures 3 and 4 show the results for *LambdaRankMAP* on all three training sets for the single-layer and 2-layer net, respectively. We see that all variations in learned weights cause a decrease in accuracy and satisfy the test for local optimality. The MAP score on the Artificial Data is higher than the MAP score on the Web datasets.

Lastly, we evaluate the local optimality of *LambdaRankMRR*. MRR training accuracy decreases as a relatively smooth function of the weights for both nets and all training sets, as shown in Figures 5 and 6, and therefore also satisfies the test for local optimality.



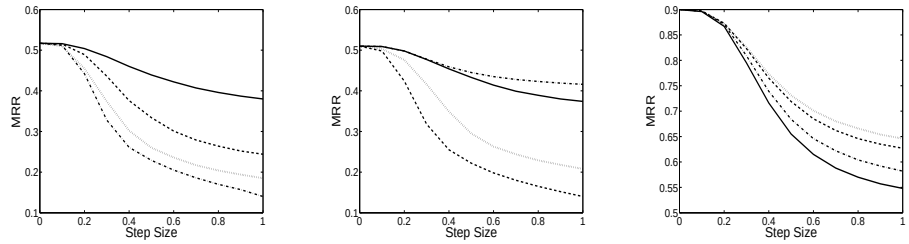
a) 10K Web Data b) 30K Web Data c) Artificial Data

Figure 3: Shifting single-layer weights for MAP@10. x-axis is the step size.



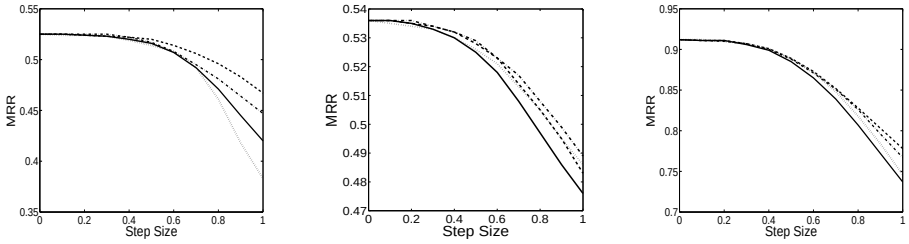
a) 10K Web Data b) 30K Web Data c) Artificial Data

Figure 4: Shifting 2-layer weights for MAP@10. x-axis is the step size.



a) 10K Web Data b) 30K Web Data c) Artificial Data

Figure 5: Shifting single-layer weights for MRR. x-axis is the step size.



a) 10K Web Data c) Artificial Data

Figure 6: Shifting 2-layer weights for MRR. x-axis is the step size.

We also verify that the training accuracy decreases for all 460 directions for all three metrics. We tested 10 steps for all directions, and found that the accuracy never increases, within a small $\epsilon = 0.003$, for both single-layer and two-layer nets trained on any of the three measures. With 99% confidence, this empirically guarantees that LambdaRank finds the locally optimum solution for NDCG, MAP, and MRR. We also verify that the gradient in fact vanishes at each learned weight by fixing all weights but one and varying the one weight under consideration. As the single weight value is varied, we again check that accuracy decreases with all variations. We find this to be true for all weights, all measures, and all training sets.

7.2 Matching Training and Target Measures

We investigate if matching the training measure to the target measure yields the best test accuracy. We evaluate the nets trained by each algorithm, *LambdaRankNDCG*, *LambdaRankMAP*, *LambdaRankMRR*, on all three IR measures. We report the results on the 10K Web data in Table 3. The left column shows the test evaluation measure. The middle columns show the measure used for training LambdaRank and the scores on the test set for the test measure. The right column is the p value resulting from a pairwise t-test. The p-value is positioned in the table between the two training measures compared in the t-test. If $p < 0.05$, then the results are statistically signigicant with 95% confidence, and are shown in bold. When testing on NDCG, training on NDCG gives the best test accuracy, with statistical significance. However, when testing on MAP, a net trained on NDCG yields similar test accuracy to a net trained on MAP. Similarly, NDCG-based training performs better than MRR-based training when testing on MRR. It appears matching the training and test measures is only best for NDCG. An immediate question arises: how training on, say, MAP can give a local optimum, but training on NDCG yields better MAP test accuracy. Recall the local optimum is found for the training set, but not the test set. Here, accuracy is reported on the test set, and the question is

Table 3: Accuracy comparison between different metrics on 10K Web Data. Bold indicates statistical significance.

Test	Train	Test Score	p value
NDCG	**NDCG**	**0.416**	
	MAP	0.412	$p < 0.05$
	MRR	0.396	$p < 0.05$
MAP	NDCG	0.442	
	MAP	0.439	$p > 0.05$
	MRR	0.429	$p < 0.05$
MRR	**NDCG**	**0.519**	
	MAP	0.516	$p < 0.05$
	MRR	0.508	$p < 0.05$

rather one of generalization, which, as we show, can be solved with more training data.

To correctly interpret these results, we consider the amount of *effective* training data. Note that MAP and MRR are both binary metrics. NDCG, on the other hand, is trained using multiple relevance levels. When the data is transformed to binary relevance, the number of training pairs decreases significantly. For the 10K Web data, the number of training pairs generated in the multilevel case is $\sim$73,000K whereas it is $\sim$27,000K for the binary case. In fact, the number of pairs actually used during training is potentially far less, especially for MRR. In Equation 11, the $\lambda-$gradient evaluates to a non-zero value only for the pairs involving a non-relevant document ranked higher than the top relevant document. For all other pairs $\{i, j\}$, λ_{ij} evaluates to zero; hence, they do not contribute to the training. MAP and MRR lose valuable information due to the binary transformation, plus MRR loses due to the calculation of the RR gain. As a result, *LambdaRankNDCG* has more training data (more learning opportunity) to make fine-grained adjustments than *LambdaRankMAP*, which has more training data than *LambdaRankMRR*.

We hypothesize that with asymptotically large amounts of effective training data, matching the training measure to the test measure will yield the best test accuracy results. To verify this claim, we repeat the analysis for 30K training queries, shown in Table 4. The 30K Web data has $\sim$60,000K pairs in the binary case, and $\sim$162,000K pairs in the multilevel case. Again, matching the training and test measure for NDCG yields the best test accuracy with statistical significance. In addition, matching the training and test measure for MRR yields the best test accuracy with statistical significance. For MAP, matching the training and test measures also gives the best performance, but without statistical significance. We

Table 4: Accuracy comparison between different metrics on 30K Web Data. Bold indicates statistical significance.

Test	Train	Test Score	p value
NDCG	**NDCG**	**0.428**	
	MAP	0.422	$p < 0.05$
	MRR	0.406	$p < 0.05$
MAP	NDCG	0.453	
	MAP	0.456	$p > 0.05$
	MRR	0.449	$p < 0.05$
MRR	NDCG	0.532	
	MAP	0.533	$p > 0.05$
	MRR	**0.537**	$p > 0.05$

predict with more queries for training, training on MAP will also produce the best accuracy since with 10K training queries, NDCG training gives the best accuracy, but with 30K queries, MAP training gives the best accuracy. As the number of queries in the training set increases, the accuracy produced by training and testing on the same measure yields the best results.

A related point is made by Robertson and Zaragoza in [8]. They claim different measures have different local optima, and likely have many local optima. They point out that a measure that responds to many flips (e.g. MAP and NDCG) will have many local optima whereas a measure that responds to fewer flips (e.g. MRR) will have fewer optima, but larger flat areas. These are the large areas of the parameter space that it cannot distinguish [8]. In other words, MAP and NDCG are more flexible than MRR, and NDCG has more granularity. Hence, it may be better to use a finer-grained metric when there is not sufficiently large training data available.

8 Conclusions

Direct optimization of IR measures has been very challenging, causing IR practitioners to build models with one (like BM25) or a few parameters that can be optimized using grid-search. However, it was recently shown that learning a model on many weak features can significantly improve test accuracy [1, 3]. In this paper, we have shown that three IR measures can in fact be optimized directly, and our $\lambda-$gradient construction is very likely extendable to other IR measures. We also show that with enough training data, matching the training measure to the target measure results in the best test accuracy. Our results open up a world of possibil-

ities for directly optimizing sophisticated models on large numbers of features for possibly any IR measure of interest. In the future, we plan to work on developing theoretical results that support our empirical findings.

References

[1] C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to rank with nonsmooth cost functions. *Neural Information Processing Systems (NIPS)*, 2006.

[2] C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to rank with nonsmooth cost functions. *Microsoft Technical Report MSR-TR-2006-60*, 2006.

[3] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *In Proc. of International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005.

[4] Z. Cao, T. Qin, T.Y. Liu, M.F. Tsai, and H. Li. Learning to rank: From pairwise to listwise approach. *In Proc. of the International Conference on Machine Learning (ICML)*, pages 129–136, 2007.

[5] K. Crammer and Y. Singer. Pranking with ranking. *In Proc. of the Conference on Neural Information Processing Systems (NIPS)*, 2001.

[6] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, pages 115–132, 2000.

[7] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Information Processing and Management*, 44(2):838–855, 2007.

[8] S. Robertson and H. Zaragoza. On rank-based effectiveness measures and optimization. *Information Processing and Management*, 10:321–339, 2007.

[9] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *Proc. of International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005.

[10] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proc. of International Conference on Machine Learning (ICML)*, 2004.

[11] J. Xu and H. Li. Adarank: A boosting algorithm for information retrieval. *In Proc. of the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 391–398, 2007.

[12] Y. Yue and C.J.C Burges. On using simultaneous perturbation stochastic approximation for ir measures, and the empirical optimality of lambdarank. *NIPS Machine Learning for Web Search Workshop*, 2007.

[13] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proc. of the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2007.