SoftRank: Optimising Non-Smooth Rank Metrics

Michael Taylor, John Guiver, Stephen Robertson and Tom Minka

Microsoft Research Cambridge

{mitaylor,joguiver,ser,minka}@microsoft.com

ABSTRACT

We address the problem of learning large complex ranking functions. Most IR applications use evaluation metrics that depend only upon the ranks of documents. However, most ranking functions generate document scores, which are sorted to produce a ranking. Hence IR metrics are innately non-smooth with respect to the scores, due to the sort. Unfortunately, many machine learning algorithms require the gradient of a training objective in order to perform the optimization of the model parameters, and because IR metrics are non-smooth, we need to find a smooth proxy objective that can be used for training. We present a new family of training objectives that are derived from the rank distributions of documents, induced by smoothed scores. We call this approach SoftRank. We focus on a smoothed approximation to Normalized Discounted Cumulative Gain (NDCG), called SoftNDCG and we compare it with three other training objectives in the recent literature. We present two main results. First, SoftRank yields a very good way of optimizing NDCG. Second, we show that it is possible to achieve state of the art test set NDCG results by optimizing a soft NDCG objective on the training set with a different discount function.

Categories and Subject Descriptors

H.3.3 [**Information Systems Applications**]:

General Terms

Keywords

learning, ranking, metrics, optimization, gradient descent

1. INTRODUCTION

There is a clear trend among both IR researchers and practitioners towards using ever more complex ranking functions. Until quite recently it has been common to use models with only a handful of free parameters. For example BM25 in its most widely adopted form has only 2. Such simple models have advantages: they are easy to tune for a given corpus, requiring few training queries and little computation to find reasonable parameter settings. Often, they work well "out-of-the-box" on new corpora with parameters reported in the literature. In short, they are robust and generalize well.

However, increasingly we see richer models appearing that set out to harness an ever expanding set of more powerful features. For example, there is much activity surrounding term proximity where work is beginning to show benefits in going beyond the bag-of-words models [13]. In the area of document structure there has been progress too. Improvements have been reported exploiting both a simple field-based flat structure, where for example, term occurrences are handled differently in titles than body text [15], and also more complex hierarchical structures in the area of XML retrieval. Much work has also been published on combining content with external cues such as link-graph features like PageRank and HITS, and more recently, usage features [1].

As these features all appear to aid retrieval effectiveness, it follows that competitive IR systems need to be able to exploit them in an efficient and reliable way. However, as the number of features increases, so does the number of parameters necessary in the ranking function. This paper addresses the problem of learning the parameters for such complex ranking functions.

The fundamental issue when formulating such a machine learning problem is the choice of *objective function* to be optimized. In IR there are very many existing metrics (NDCG, MAP, RPrec etc.) that all share the property that they are rank-dependent, placing more emphasis on performance at the top of a list of documents, thus reflecting the end-user experience. While these metrics are ideal for *evaluating* trained systems, their use as objective functions for *training* is problematic.

1.1 IR Metrics are Not Smooth

Given a representative set of queries and relevance judgments, we seek to learn a ranking function (or model) that takes a set of document-query match feature values and generates a score. At test time, this score is used to sort documents to produce a ranked list, which is to be evaluated using an IR metric.

Following [4, 3] in this paper we choose to model the mapping from features to score using a 2-layer neural net model. Neural nets represent a tried-and-tested machine learning technique that scales well with large amounts of training data. The optimization process used is gradient based, and

Most of this material was presented at SIGIR LETOR Workshop '07 Amsterdam, Netherlands

so this learning approach depends upon the availability of a gradient of the training objective.

Typical IR metrics only depend on the ranks (and not the scores). If we make small changes to the model parameters, the scores will typically change smoothly, but the ranks of documents will not change until one document's score passes another, at which point the IR metric will make a discontinuous change (see Figure 1). In other words, the IR metrics are non-smooth with respect to model parameters: they are everywhere either flat (with zero gradient) or discontinuous.

Section 2 describes our chosen test metric NDCG, and then gives a brief description of previous work in constructing useful training utilities that have smooth gradients and also attempt to approximate the desired non-smooth rank-dependent IR objective. Section 3 presents a new training objective called SoftNDCG that is a smoothed approximation to NDCG.

2. RANKING UTILITIES

This section first describes the test metric that we adopt in this paper, called Normalized Discounted Cumulative Gain (NDCG). The approach we adopt in this paper is to seek an objective function that makes a good proxy for a rank-based metric (e.g. NDCG), but that is also differentiable with respect to the parameters of the ranking function. Such a function will be more easily optimized in the neural net framework. We next describe several training utilities that have been previously proposed, concentrating on RankNet [4] and LambdaRank [3] as they are used as baselines in this paper.

For a given training query, we will assume we have N documents, each with a known human-defined rating. We denote an individual document indexed by j as doc_j. Let us assume that we have a ranking function that takes in document features $\mathbf{x}_j$ and produces a score s_j. In this paper we consider the class of non-linear ranking functions (models) represented by the 2-layer neural net f with weights (parameters) $\mathbf{w}$. We denote the score:

$$s_j = f(\mathbf{w}, \mathbf{x}_j). \tag{1}$$

2.1 NDCG

We adopt the IR evaluation metric called NDCG [10] because it is a reasonable way of dealing with multiple relevance levels in our datasets. It is often truncated at a rank position R (indexed from 0) and is defined as:

$$G_R = G_{R,max}^{-1} \sum_{r=0}^{R-1} g(r) D(r) \tag{2}$$

where the *gain* $g(r)$ of the document at rank r is usually an exponential function $g(r) = 2^{l(r)}$ of the labels $l(r)$ (or *rating*) of the document at rank r. The labels typically take values from 0 (bad) to 4 (perfect). A popular choice for the rank discount is $D(r) = 1/\log(2+r)$ and $G_{R,max}$ is maximum value of $\sum_{r=0}^{R-1} g(r)D(r)$ obtained when the documents are optimally ordered by decreasing label value. Where no subscript is defined, it should be assumed that $R = N$.

2.2 Pointwise

The simplest class of smooth training objective is what we call a pointwise objective because it can be computed for a single document. If we are given a target label for each document, then the most straightforward pointwise objective is the mean squared error (MSE) between the target and predicted label. We can write the objective for a document/label pair (in this case it is actually a cost) as:

$$U_{\text{MSE}}(s_j) = (s_j - l_j)^2 \tag{3}$$

and the total cost is obtained by computing the mean over all training documents. This is the first baseline objective that we will use in our experiments. Another option would be to treat the NDCG *gains* as the targets for the regression, instead of the labels. This made little difference in our experiments and we do not consider it further.

There are of course more sophisticated approaches to pointwise metrics. Rankprop [6] alternates between a MSE approach described above, and a phase that iteratively adjusts the targets themselves. An alternative approach is the framework of ordinal regression, which is closer to classification in spirit. An example of this approach is PRank [8]; for a Bayesian treatment see [7].

2.3 Pairwise

Because ranking metrics only require the recovery of *relative* relevance levels, the motivation of this approach is that pairwise preferences in the labels may well be more easily modelled than any available absolute value of relevance. Herbrich *et. al.* [9] were the first to use pairwise preference labels to learn ranking functions. They took an ordinal regression approach using an SVM model. This model is well known in the IR literature as the RankingSVM [11], where it was used in a scenario where only preference labels were available in the training data, derived from click-through logs.

RankNet [4] is a probabilistic model for pairwise preferences. The algorithm assumes that it is provided with a training set consisting of pairs of documents doc_1, doc_2 together with a target probability $\bar{P}_{12}$ that doc_1 is to be ranked higher than doc_2. The authors define a ranking function f as we have specified in (1).

The map from the outputs s_j to probabilities is modeled using a logistic function $P_{12} \equiv e^Y/(1+e^Y)$ where $Y \equiv s_2 - s_1$, and P_{12} is the probability that doc_1 is ranked higher than doc_2. They then invoke the cross-entropy error function to penalize pair ordering errors:

$$U_{\text{RN}}(Y) = -\bar{P}_{12} \log P_{12} - (1-\bar{P}_{12}) \log(1-P_{12}). \tag{4}$$

This is a very general cost function that affords the use of any available uncertainty we may have concerning the pairwise ratings. Following [4], in our implementation of RankNet, we take the pair ordering data as certain (ignoring ties), and so for us, the $\bar{P}_{12}$ are always one. With this simplification, the RankNet cost for a pair becomes:

$$U_{\text{RN}} = \log\left(1 + e^{s_2 - s_1}\right) \tag{5}$$

where the score difference is positive if the documents are in the wrong order. The RankNet cost for a pair in the wrong order tends to a linear fuction of score difference. When the scores are correctly ordered, U_{RN} asymptotically approaches zero as the score difference increases. Thus the gradient of the U_{RN} not only encourages the pair to be in the right order, but encourages them to have at least some separation in their scores. If f is differentiable with respect to the parameters, then so too will U_{RN}.

2.4 Rank Dependent

The smooth training utilities described so far do not take into account the rank of a document in the set of documents scored for a training query. Hence, Burges *et. al.* [3] argue that there is a possibility that a model might be prone to waste capacity in improving the order of documents at low (poor) ranks at the expense of documents at the top of the ranking. Their approach, called LambdaRank, argues for a training objective that is closer to NDCG, in that it should care more about the top of the ranking than the bottom. Consequently, it needs to incorporate rank information into its training objective.

However, rank information is only directly available via a sort, and the sort operation is inherently undifferentiable. Thus, any sort operation makes the objective non-smooth. They get round this problem by defining a virtual gradient on each document *after* the sort. In other words, LambdaRank defines a gradient of an *implicit* objective function that is itself never actually defined. These gradients are only defined once the current model has produced a ranked list for the query. For example, consider a query with just two relevant documents doc_1, doc_2, and at some stage in training the model places doc_1 near the top of the ranked list with score s_1 and doc_2 near the bottom with score s_2. The intuition concerning limited capacity can be encoded as:

$$\left|\frac{\partial U}{\partial s_1}\right| \gg \left|\frac{\partial U}{\partial s_2}\right|, \tag{6}$$

or in words, we would like the rate of change of the objective with respect to the high-ranking relevant document's score to be very much greater than that for the low-ranking relevant document. Note that U is the implicit objective that is not defined. Only the gradients of U are defined, given a sorted list of documents at a particular point in training. In their paper, they report that they tried many different gradient functions satisfying the capacity constraint of (6), and the one that worked best on web retrieval experiments was as follows. Given a pair of documents for a training query, they define the gradient (lambda function) to be the gradient of the RankNet cost (5) scaled by the difference in NDCG found by swapping the two documents in question. Using this pairwise "force", the total gradient for a document j can be obtained by summing all such pairwise interactions, giving:

$$\lambda_j \equiv \frac{\partial U_{\text{LR}}}{\partial s_j} = G_{max}^{-1} \sum_i \left(\frac{1}{1+e^{s_i - s_j}}\right)(g_i - g_j)\left(D_i - D_j\right) \tag{7}$$

where g_i is the gain of the label of doc_i and D_i is the discount at the rank of doc_i, $D(r_i)$. The authors report that a 2-layer neural net model trained using this LambdaRank objective outperforms the same model using the RankNet cost.

A more recent approach [5] sets out to define a probability distribution over rankings: in other words, the event space consists of document permutations. To contrast that work with this paper, here we only set out to elicit distributions for the ranks of individual documents: an event in our space is that of an individual document having a particular rank. The probability of such an event is obtainable from a full ranking distribution by summing up the probabilities of the rankings for which this event occurs; however, in our work, we deal directly with the simpler event space.

3. SOFTRANK

The approach taken by LambdaRank was to abandon the attempt to define an explicit smooth objective, and instead only work with an implicit objective via the definition of gradient functions with intuitively desirable properties.

The main idea that motivates the SoftRank approach is the observation that if we consider the scores to be smoothed by treating them as random variables, then it should be possible to propagate that noise through to a rank-dependent IR metric. In particular, in this paper we use this idea to create a smoothed approximation to NDCG (referred to as SoftNDCG), but the approach could equally be applied to other rank-based metrics. This section shows under what assumptions we can write down an analytic expression for the SoftNDCG and its derivative with respect to the model parameters $\mathbf{w}$. The process is summarized as a factor graph in Figure 4, the components of which are discussed in the following sections.

In order to make an objective dependent upon the ranks of documents, it is natural to assume that a sort is required at some stage. However, this would render the objective non-differentiable, so our approach is based on the idea that we need to avoid sorting.

At a high level, the approach we adopt is as follows. We consider N labeled documents for a single query. This means we have N score distributions (Section 3.1). We show how we can map from score distributions to a rank distribution for each document (see Section 3.2) without performing an explicit sort. Armed with a rank distribution for each document, we investigate under what conditions we can compute an *expected* NDCG (Section 3.3). The expected smoothed NDCG is what we call SoftNDCG. Finally, because none of these steps necessarily involves a sorting operation, we show that the expression for SoftNDCG can be differentiated with respect to the model parameters, and thus show how it can be optimized using gradient ascent (Section 3.4).

We conclude the description of SoftRank by discussing how the single degree of freedom represented by the global scale of the scores is handled (Section 3.5), and finally discuss computational issues (Section 3.6).

3.1 Smoothing Scores

Rather than representing scores as deterministic values, we will treat them as smoothed score distributions (see Figure 2). The simplest way to do this, and the approach we adopt for the remainder of this paper, is to give every score the same smoothing using equal variance Gaussian distributions. Hence the deterministic score s_j in (1) becomes the mean of a Gaussian score distribution[1], with a shared smoothing variance σ_s:

$$p(s_j) = \mathcal{N}(s_j|\bar{s}_j, \sigma_s^2) = \mathcal{N}(s_j|f(\mathbf{w}, \mathbf{x}_j), \sigma_s^2). \tag{8}$$

An alternative motivation would be to consider the source of noise as an inherent uncertainty in the model parameters $\mathbf{w}$, arising from inconsistency between the ranking model and the training data (see the left side of Figure 4). This would be the natural result of a Bayesian approach to the learning task. Mapping parameter distributions through the non-linearities of a 2-layer neural net that we use in this paper is not straightforward, so we do not pursue this idea here.

[1]Using $\mathcal{N}(x|\mu, \sigma^2) = (2\pi\sigma^2)^{-0.5} \exp[-(x-\mu)^2/2\sigma^2]$

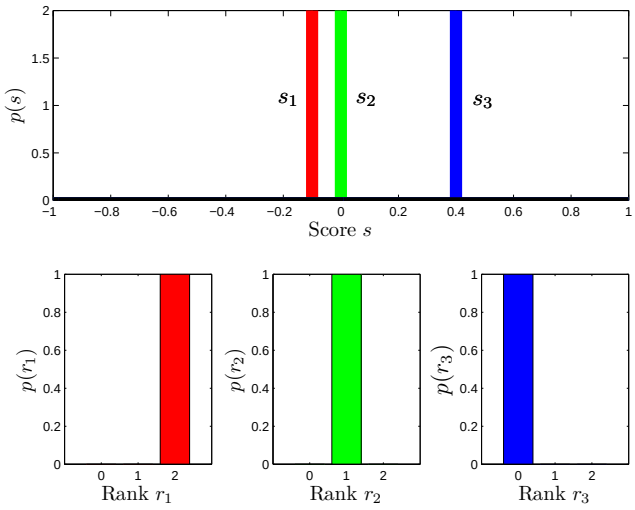

Figure 1: Deterministic scores and ranks: *Three document scores as point (deterministic) values and their corresponding rank distributions. The lowest scoring document s_1 is certain to be ranked in the lowest position 2.*

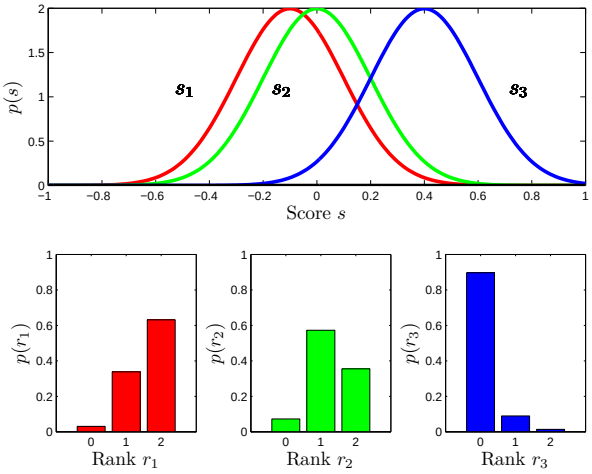

Figure 2: From score to rank distributions: *Smoothed scores for 3 documents and the resulting 3 rank distributions.*

3.2 From Score to Rank Distributions

When we have deterministic scores, we have deterministic rank distributions, as shown in Figure 1. This section presents a way of quantifying what happens to the rank distributions when we add noise to (smooth) the score distributions, shown in Figure 2.

The rank distributions shown in Figure 2 may be simulated by the following exact generative process: a) sample a vector of N scores, one from each score distribution, b) *sort* the score samples and c) accumulate histograms of the resulting ranks for each document. However, we wish to avoid the sort so we can perform gradient based optimization, and so the remainder of this section presents an approximate algorithm for generating the rank distributions that avoids an explicit sort.

For a given doc_j, consider the probability that another doc_i will rank above doc_j. Denoting S_j as a draw from $p(s_j)$, we require the probability that $S_i > S_j$, or equivalently $\Pr(S_i - S_j > 0)$. Therefore the required probability is simply the integral of the difference of two Gaussian random variables, which is itself a Gaussian [14], and therefore the probability that document i beats document j, which we will henceforth refer to as π_{ij}, is:

$$\pi_{ij} \equiv \Pr(S_i - S_j > 0) = \int_0^\infty \mathcal{N}(s|\bar{s}_i - \bar{s}_j, 2\sigma_s^2)ds. \quad (9)$$

This quantity represents the fractional number of times we would expect doc_i to rank higher than doc_j on repeated pairwise samplings from the two Gaussian score distributions. For example, in Figure 2, we would expect $\pi_{32} > \pi_{12}$. In other words, if we were to draw two pairs $\{S_1, S_2\}$ and $\{S_3, S_2\}$, S_3 is more likely to win its pairwise contest against S_2 than S_1 is in its contest.

Now we use these pairwise probabilities to generate ranks. We argue intuitively that if we were to add up the probabilities of a document being beaten by each of the other documents, then we would have a quantity that is related to the expected rank of the document being beaten. In other words, if a document is never beaten, its rank will be 0, the best rank. More generally, using the pairwise contest trick, we can write down an expression for the expected rank r_j of document j as:

$$E[r_j] = \sum_{i=1, i\neq j}^{N} \pi_{ij} \quad (10)$$

which can be easily computed using (9). As an example, Figure 2 shows what happens to the rank distributions when we smooth the scores: the expected rank of document 3 is between 0 (best) and 1, and documents 2 and 3 have an expected rank between 2 and 3 (worst).

The actual *distribution* of the rank r_j of a document j under the pairwise contest approximation is obtained by considering the rank r_j as a Binomial-like random variable, equal to the number of successes of $N-1$ Bernoulli trials, where the probability of success is the probability that document j *is beaten by* another document i, namely π_{ij}. If i beats j then then r_j goes up by one.

However, because the probability of success is different for each trial, it is a more complex discrete distribution than the Binomial: we call it the *Rank-Binomial* distribution. Like the Binomial, it has a combinatoric flavour: there are few ways that a document can end up with top (and bottom) rank, and many ways of ranking in the middle. Unlike the Binomial, it does not have an analytic form. However, it can be computed using a standard result from basic probability theory, that the probability density function (pdf) of a sum of independent random variables is the convolution of the individual pdfs [14]. In this case we have a sum of N independent Bernoulli (coin-flip) distributions, each with a probability of success π_{ij}. This yields an exact recursive computation for the distribution of ranks as follows.

If we define the initial rank distribution for document j as $p_j^{(1)}(r)$, where we have just the document j, then the rank can only have value zero (the best rank) with probability

one:

$$p_j^{(1)}(r) = \delta(r) \tag{11}$$

where $\delta(x) = 1$ only when $x = 0$ and zero otherwise. Now we have $N-1$ *other* documents that contribute to the rank distribution that we will index with $i = 2..N$. Each time we add a new document i, the event space of the rank distribution gets one larger, taking the r variable to a maximum of $N-1$ on the last iteration. The new distribution over the ranks is updated by applying the convolution process described above, giving the following recursive relation:

$$p_j^{(i)}(r) = p_j^{(i-1)}(r-1)\pi_{ij} + p_j^{(i-1)}(r)(1-\pi_{ij}). \tag{12}$$

This can be interpreted in the following more intuitive manner. If we add document i, we can write the probability of rank r_j as a sum of two parts corresponding to the new document i beating document j or not. If i beats j then the probability of being in rank r at this iteration is equal to the probability of being in rank $r-1$ on the previous iteration, and we have the situation covered by the first term on the right of (12). Conversely, if the new document leaves the rank of j unchanged (it loses), the probability of being in rank r is the same as it was in the last iteration, corresponding to the second term on the right of (12).

We note that we need to define $p_j^{(i)}(r) = 0$ if $r_j < 0$. We define the final rank distribution $p_j(r) \equiv p_j^{(N)}(r)$. Figure 2 shows these distributions for the simple 3 score case.

To conclude this section, we note that the pairwise contest trick yields Rank-Binomial rank distributions, which are an approximation to the true rank distributions. Their computation does not require an explicit sort. Simulations have shown that this gives similar rank distributions to the true generative process. We can improve these approximations further by performing a sequence of column and row operations on the $[p_j(r)]$ matrix: divide each column by the column sums, then divide each row of the resulting matrix by the row sums, and iterate to convergence. This process is known as Sinkhorn scaling, its purpose being to convert the original matrix to a doubly-stochastic matrix. The solution can be shown to minimize the Kullback-Leibler distance of the scaled matrix from the original matrix [2]. We will show later in our results that we can successfully optimize NDCG using these approximate rank distributions, which further justifies the pairwise independence approximation and the Sinkhorn post-processing.

3.3 SoftNDCG

This section shows how we can use rank distributions to smooth traditional IR metrics. The general approach is to take the expectation of the IR metric with respect to the rank distribution. As an example, we will now examine the specific case of NDCG.

The expression for deterministic NDCG was given in (2) as $G = G_{max}^{-1} \sum_{r=0}^{N-1} g(r)D(r)$. We set out to compute the *expected* NDCG given the rank distributions described in the last section. Rewriting NDCG as a sum over document indices rather than document ranks we get:

$$G = G_{max}^{-1} \sum_{j=1}^{N} g(j)D(r_j). \tag{13}$$

With reference to Figure 3 we replace the deterministic discount $D(r)$ with the expected discount. Thus we define soft NDCG $\mathcal{G}$ as:

$$\mathcal{G} = G_{max}^{-1} \sum_{j=1}^{N} g(j)E[D(r_j)]. \tag{14}$$

The expected discount is obtained by mapping the rank distribution through the non-linear deterministic discount function (again, see Figure 3) to give:

$$\mathcal{G} = G_{max}^{-1} \sum_{j=1}^{N} g(j) \sum_{r=0}^{N-1} D(r)p_j(r) \tag{15}$$

where the rank distribution $p_j(r)$ is given in (12).

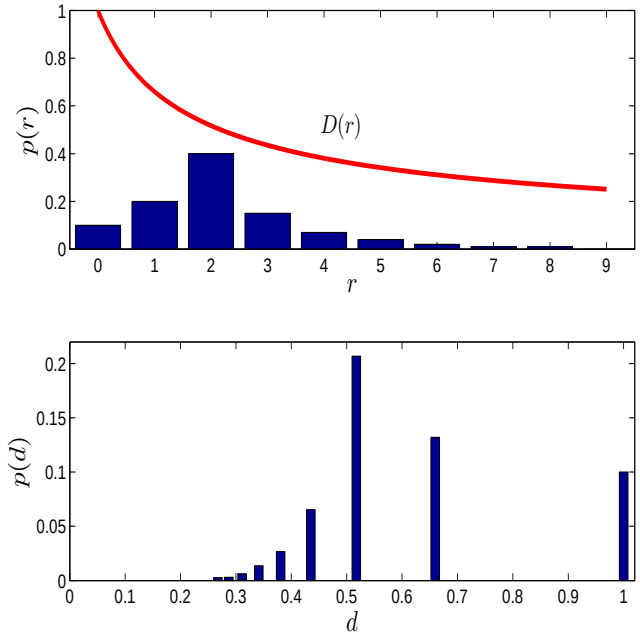

Figure 3: From rank to discount distribution. *The rank distribution is mapped through the non-linear discount function D, to give a discrete distribution over discounts $p(d)$ whose expectation we substitute for the deterministic discount to obtain SoftNDCG.*

3.4 Gradient of SoftNDCG

Having derived an expression for a SoftNDCG, we now differentiate it with respect to the weight vector. The derivative with respect to the weight vector with K elements is:

$$\frac{\partial \mathcal{G}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial s_1}{\partial w_1} & \cdots & \frac{\partial s_N}{\partial w_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial s_1}{\partial w_K} & \cdots & \frac{\partial s_N}{\partial w_K} \end{bmatrix} \begin{bmatrix} \frac{\partial \mathcal{G}}{\partial \bar{s}_1} \\ \cdots \\ \frac{\partial \mathcal{G}}{\partial \bar{s}_N} \end{bmatrix}. \tag{16}$$

The first matrix is defined by the neural net model and is computed via backpropagation [12]. The second vector is the gradient of our objective with respect to the score means. As with LambdaRank above, our task is to define this gradient vector for each document in a training query.

Taking a single element of this gradient vector corresponding to a document with index m ($1 \le m \le N$), we can

differentiate (15) to obtain:

$$\frac{\partial \mathcal{G}}{\partial \bar{s}_m} = G_{max}^{-1} \sum_{j=1}^{N} g(j) \sum_{r=0}^{N-1} D(r) \frac{\partial p_j(r)}{\partial \bar{s}_m}. \quad (17)$$

Intuitively, this says that changing score $\bar{s}_m$ affects $\mathcal{G}$ via potentially all the rank distributions, as moving a score will affect every document's rank distribution. The resultant change in each rank distribution will induce a change in the expected gain for each document determined by the non-linear discount function $D(r)$.

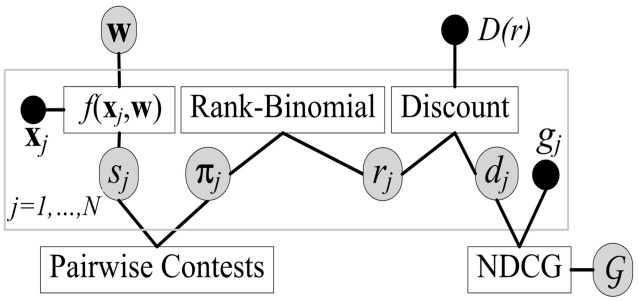

Figure 4: A factor graph of the distributions for a query. *In our development of SoftRank to date, we have only worked with Gaussian scores s_j. These map to Bernoulli vectors $\boldsymbol{\pi}_j$ which provide the success probabilities for the computation of the Rank-Binomials over ranks r_j for each document 1..N (see Section 3.2). Then the rank distributions get mapped in a non-linear way through the discount function $D(r)$ to give a distribution over discounts d_j (see Section 3.3). Finally, combining the expected discount with the gain of the label over all documents, we arrive at the expected SoftNDCG.*

Hence we need a parallel recursive computation to obtain the required derivative of $p_j(r)$. Denoting $\psi_{m,j}^{(i)}(r) = \frac{\partial p_j^{(i)}(r)}{\partial \bar{s}_m}$ it is easy to show from (12) that:

$$\begin{aligned} \psi_{m,j}^{(1)}(0) &= 0 \\ \psi_{m,j}^{(i)}(r) &= \psi_{m,j}^{(i-1)}(r-1)\pi_{ij} + \psi_{m,j}^{(i)}(r)(1-\pi_{ij}) \\ &\quad + \Big(p_j^{(i-1)}(r-1) - p_j^{(i-1)}(r)\Big)\frac{\partial \pi_{ij}}{\partial \bar{s}_m} \quad (18) \end{aligned}$$

where again the recursive process runs $i = 1..N$. Considering now the last term on the right of (18), differentiating π_{ij} with respect to $\bar{s}_m$ using (9) yields three different cases, given we know already that $i \neq j$ and so $m = i = j$ is not possible). Using the fact that

$$\frac{\partial}{\partial \mu} \int_0^\infty \mathcal{N}(x|\mu,\sigma^2)dx = \mathcal{N}(0|\mu,\sigma^2) \quad (19)$$

it can easily be shown that from (9) that:

$$\frac{\partial \pi_{ij}}{\partial \bar{s}_m} = \begin{cases} \mathcal{N}(0|\bar{s}_m - \bar{s}_j, 2\sigma_s^2) & m = i, m \neq j \\ -\mathcal{N}(0|\bar{s}_i - \bar{s}_m, 2\sigma_s^2) & m \neq i, m = j \\ 0 & m \neq i, m \neq j \end{cases} \quad (20)$$

and so substituting (20) in (18), we can now run the recursion for the derivatives. We define the result of this computation as the N-vector over ranks:

$$\frac{\partial p_j(r)}{\partial \bar{s}_m} \equiv \boldsymbol{\psi}_{m,j} = [\psi_{m,j}^{(N)}(0), ..., \psi_{m,j}^{(N)}(N-1)]. \quad (21)$$

Using this matrix notation we substitute the result in (17):

$$\frac{\partial \mathcal{G}}{\partial \bar{s}_m} = \frac{1}{G_{max}} [g_1, ..., g_N] \begin{bmatrix} \boldsymbol{\psi}_{m,0} \\ ... \\ \boldsymbol{\psi}_{m,N-1} \end{bmatrix} \begin{bmatrix} d_0 \\ ... \\ d_{N-1} \end{bmatrix}. \quad (22)$$

We now define the gain vector $\mathbf{g}$ (by document), the discount vector $\mathbf{d}$ (by rank) and the $N \times N$ square matrix Ψ_m whose rows are the rank distribution derivatives implied above:

$$\frac{\partial \mathcal{G}}{\partial \bar{s}_m} = \frac{1}{G_{max}} \mathbf{g}^\mathsf{T} \Psi_m \mathbf{d}. \quad (23)$$

So to compute the N-vector gradient of $\mathcal{G}$ which we define as $\nabla \mathcal{G} = [\frac{\partial \mathcal{G}}{\partial \bar{s}_1}, ..., \frac{\partial \mathcal{G}}{\partial \bar{s}_N}]$ we need to compute Ψ_m for each document.

3.5 Scale Optimizations

Any optimization of an objective that is based on ranks of documents should consider carefully the degree of freedom that corresponds to an arbitrary scaling of the scores. Multiplying all the scores by a common factor does not affect the ranks, and if it is not handled properly, could lead to an undesirable degeneracy in the optimization process.

With SoftRank, this global scale factor is equivalent to the score variance σ_s^2: if we multiply all the scores by some common factor, it is the same thing as dividing σ_s by that factor. In the experiments described in this paper, σ_s is set to some initial value, and is not changed during optimisation. Hence the scale factor is controlled by $\nabla \mathcal{G} \cdot \bar{\mathbf{s}}$, the component of the SoftNDCG gradient parallel to the current mean score vector $\bar{\mathbf{s}}$.

As a simple example, imagine a query with 2 documents that were stubbornly in the wrong order, say the relevant document was always below the irrelevant. In this situation, SoftRank would steadily decrease the scale which would have the effect of making the Gaussians overlap more, thus increasing the probability that the relevant document could beat the irrelevant above it. Conversely, if the documents were correctly ordered, the scale would increase, effectively reducing the score variance.

Following this rather intuitive argument, we conclude that SoftRank does control the scale of the scores in a sensible, non-degenerate way. In effect it overloads the global scale degree of freedom to implement an annealing schedule. It is the subject of ongoing study as to whether this default schedule can be improved upon.

3.6 Computational Considerations

For a given query of N documents, calculation of the π_{ij} is $O(N^2)$, calculation of all the $p_j(r)$ is $O(N^3)$, and calculation of the SoftNDCG is $O(N^2)$. Similar complexity arises for the gradient calculations. So the calculations are dominated by the recursions in (12) and (18).

A substantial computational saving can be made by approximating all but a few of the Rank-Binomial distributions. The motivation for this is that a true binomial distribution, with N samples and probability of success π, can be approximated by a normal distribution with mean $N\pi$ and variance $N\pi(1-\pi)$ when $N\pi$ is large enough. For the rank binomial distribution, π is not constant, but simulations

confirm that it can be approximated similarly, for a given j, by a normal distribution with mean equal to the expected rank $\sum_{i=1,i\neq j}^{N} \pi_{ij}$ and variance equal to $\sum_{i=1,i\neq j}^{N} \pi_{ij}(1-\pi_{ij})$. As the approximation is an explicit function of the π_{ij}, we can easily calculate the gradients of the approximated $p_j(r)$ with respect to π_{ij}, and therefore with respect to the $\bar{s}_m$. Using this approximation allows us to restrict the expensive recursive calculations to a few documents at the top and bottom of the ranking.

4. EXPERIMENTS

We performed experiments on three corpora: queries from the TREC .GOV corpus, on enterprise search data and queries from a commercial web search engine. For all corpora, we used only those documents containing *all* the query terms (the AND set) for both training and testing. This simplification is consistent with common practice for large scale commercial search engines.

4.1 Training Set Subsampling

To make training tractable, it is necessary to subsample the irrelevant documents to some degree. If we do not do this, the data we need to process is dominated unnecessarily by uninformative irrelevant documents. In this paper, we adopted a very simple approach. We took all the judged documents, whatever the relevance label. We then augmented this set with documents drawn at random from the AND set of unlabeled documents, which we assume to be irrelevant, until we have drawn a number equal to the size of the judged set, or we have drawn 30, whichever happens first. This threshold is admittedly rather arbitrary, and it is the subject of ongoing experiments as to how sensitive each objective is to this number.

4.2 .GOV

For the TREC .GOV corpus, we took the following sets of topics: name/home page 2003 (300 queries), topic distillation 2003 (50 queries) and all queries from 2004 (225 queries). As mentioned above, we chose to work with AND set documents only, so we dropped all queries with no relevant documents in the AND set, and this left us with 508 queries. This number would have been greater if we had performed stemming. The queries had binary relevance judgments.

The data was partitioned for 5-fold (3:1:1) cross validation, giving 5 runs, each with roughly 300 training queries, 100 validation and 100 test. It is possible that a different ratio of train/validate/test queries might give better results, and again this is an area of current investigation.

The index of our crawl of .GOV has 3 structural fields: body, title and anchor. We used BM25F, which in this case has 6 tuneable parameters that can be learned using back-propagation following the process described in [16]. In addition, we used PageRank as a further feature.

These two features were the input to a 2-layer net with 3 hidden nodes. This number was obtained by trying a few values for the LambdaRank baseline. No further effort was made to find the best number for each individual objective. No linear model (single layer net) results are reported here, as it is well-established in the literature now [3] that they do not perform as well as non-linear models. The training queries were subsampled as described in section 4.1. Complete AND sets were used for validation and test.

4.3 Enterprise Search

This data came from the Intranet of a large corporation with about 15 million documents. Queries were sampled from the existing search service log. Judges were asked to choose queries they understood and assign one of 4 relevance levels. We used 761 queries and performed 5-fold cross validation here with the same train/validate/test ratios as above. The index had 6 fields, including author and url, in addition to the body, title and anchor used for .GOV. We used the 12-parameter BM25F, alongside about 8 query independent features, including file type and url length. We used a 2-layer net with 4 hidden nodes. Training, validation and test queries were sub-sampled as described in section 4.1.

4.4 Web Search Data

This data came from a large commercial web search engine. We use a 4096 training, 2651 validation and 2560 test queries, sampled from the live query log, with 5 relevance levels. In this experiment we used about 380 features. Because we had more features and training queries, we used a more complex 2-layer model with 10 hidden nodes. The training queries were sub-sampled as described in section 4.1. While complete AND sets were not used for validation and test, these splits still had about 20 times as many randomly sampled documents from the AND set, which were assumed to be irrelevant.

4.5 Optimization

For each run we initialize the weights to a random setting near zero, so that all nodes in the neural net start off unsaturated. Following standard neural net practice, all input features are normalized so that they are zero mean and unit standard deviation over the training set queries. We call a cycle through all training queries an *epoch*. If the training set NDCG@10 (G_{10}) goes for 16 epochs without increasing, we reinitialize the weights. Our runs were for 128 epochs.

Like [3] we adopted a stochastic gradient descent approach, where the weights of the model were updated in a batch mode after each query. This optimization technique is very simple. There is one parameter that corresponds to the distance taken along the gradient vector at each weight update, called the learning rate. We set this rate to an initial value, and we reduce it by a factor of 0.8 each time the end-of-epoch training set G_{10} does not improve.

Initial values for both the learning rate (all utilities) and the SoftRank smoothing σ_s affect the results significantly. Therefore, all experiments need to set these values using the validation set. Multiple training runs are performed from different initial settings, and the run/epoch that performed best on the validation set is the one used in the final evaluation on the test set. We tried learning rates from 10^{-1} to 10^{-7} and initial smoothing from 10^{0} to 10^{-4}.

For consistency across these experiments, we have only made use of gradient information. However, the SoftRank approach supports future incorporation of rank-based metrics directly into the optimization process which opens up the possibility of using more sophisticated optimizers which do not rely on gradient alone, such as BFGS. This is the subject of future work.

4.6 Initial Results

We investigated 4 utilities on the three corpora: MSE (3),

RankNet (RN) (5), LambdaRank (LR) (7) and SoftRank (SR) (23). In this set of initial experiments we used the same discount function for the training objectives (SR and LR) as that used in the test NDCG defined in Section 2.1.

The mean NDCG over all test set queries, at cut-offs 3 and 10 (G_3, G_{10}) are shown in Table 1. We use the paired t-test at the 5% significance level on the G_{10} results only, as this was the objective used for model selection on the validation set.

	.GOV		Enterprise		Web	
	G_3	G_{10}	G_3	G_{10}	G_3	G_{10}
MSE	56.0	59.5	59.0	62.0	60.7	65.8
RankNet	65.4	67.7	58.1	61.9	60.6	65.4
LambdaRank	65.9	68.1	59.5	62.5	61.4	66.4
SoftRank	66.9	68.9	59.3	62.6	60.4	65.6

Table 1: Test set NDCG@10 and NDCG@3 for the four utilities and three corpora.

.GOV

Here the MSE objective performed surprisingly badly, and significantly worse than the others. Taking the RN result as the baseline, LR did not do better ($p = 55\%$) and SR was significantly better ($p = 4\%$). SR was close to being significantly better than LR ($p = 8\%$).

Enterprise

Here the MSE objective performed surprisingly well, being equivalent to RN. LR was not significantly better than MSE, though nearly so at $p = 7\%$ and SR>MSE at $p = 2\%$. Finally SR was not significantly better than LR.

Web

In this experiment, characterized by a very complex model with several thousand parameters, we found that LambdaRank was the best on the test set, significantly beating all other utilities. MSE did surprisingly well. MSE, RN and SR were all not significantly different.

	Training G_{10}
MSE	67.9
RankNet	67.6
LambdaRank	69.2
SoftRank	70.6

Table 2: *Training set NDCG@10 for the four objectives on the web corpus.*

Training Set G_{10}

Table 2 compares the *training* set G_{10} values. As expected, we observe that LR and SR are both much better than MSE and RN. We also note that SR yields a consistently and significantly better fit to the training data. This effect was observed on all three corpora. We conclude that gradient ascent on SoftNDCG represents a more effective NDCG optimization algorithm than the other objectives.

We find it encouraging that, using the same models and gradient-based optimizer, SoftRank consistently finds better training set G_{10} values. In this, SoftNDCG has fulfilled the goal of creating a smoothed approximation of a rank-based metric such as NDCG, and optimizing it directly. It seems therefore that the Rank-Binomial approximation is a good one.

4.7 Generalization Investigation on Web Corpus

However, further comment is needed on test set performance. The training set G_{10} values lead us to ponder how SR can give a much better fit to the training data at the same time as worse test set performance given that the model structures are identical.

Simpler Linear Model

Our first thought was that we might just be overfitting: maybe LambdaRank has better natural regularization properties than SoftNDCG, and the use of early stopping on the validation set is not sufficient for SR. To test this hypothesis we tried a drastically simpler linear model, reducing the number of model parameters by a factor of 10 from about 4000 to 400. This simple model could not realistically be overfitted with 4K training queries.

Model	Train G_{10}	Test G_{10}
2-layer LR	69.2	66.4
2-layer SR	70.6	65.7
Linear LR	67.2	65.2
Linear SR	67.5	64.6

Table 3: *Results on a much simpler model: the simple linear model still shows a gain for LambdaRank.*

The results of this experiment are shown in Table 3. As we would expect, the training set NDCGs are worse for the linear model than for the non-linear model as the model is less flexible. More interestingly, we still observe that LR statistically significantly outperforms SR in the linear model on the test set. Now given we are very unlikely to be under-regularized for the reasons stated earlier, this leads us to conclude that we are not overfitting in the classical sense.

Alternative training discount functions

Another possible explanation is that by using the NDCG discount function for training, we have allowed the model to concentrate *too much* on high-ranking documents in the training set. In other words, perhaps there is useful information in the ordering of documents beyond the top 10 or so relevant documents that SoftNDCG is effectively ignoring, which the baseline models do not. Therefore, maybe the use of a less severe *training* discount function would allow SoftRank to generalize better to new queries, by exploiting more of the training data.

To test this hypothesis we have investigated a variety of training discount functions that do not decay as quickly as the regular NDCG discount function. These are shown in Figure 5, ranging from convex (super-linear with rank, and denoted $\alpha = -1$ in Figure 6), through linear $\alpha = 0$, to concave (sub-linear, like the regular NDCG discount) $\alpha = 1$.

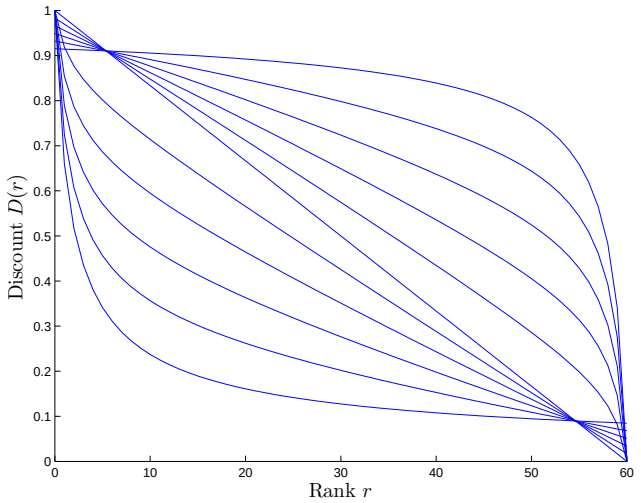

Figure 5: Shallower discount functions: *We tried a range of discount functions ranging from super-linear (top, $\alpha = -1$) through linear ($\alpha = 0$) to sub-linear, as used in the test NDCG (bottom, $\alpha = 1$).*

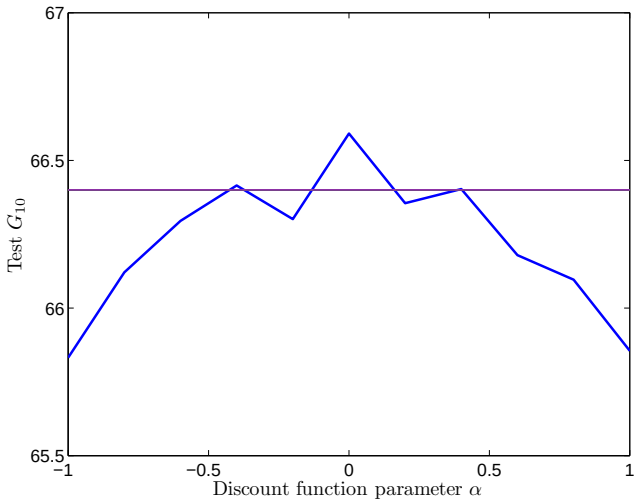

Figure 6: Test set performance with different discount functions *Test set G_{10} on web data against the discount function parameter α as defined in Figure 5. The horizontal line is the LR baseline. SR does as well as LR for $-0.4 < \alpha < 0.4$.*

We used these new, deeper discount functions for both training (in SoftNDCG) and validation NDCG, but retaining the original discount for the test set.

The results of this experiment are shown in Figure 6. For $\alpha = 1$ we saw a slightly (not significant) better G_{10} as we are using a validation NDCG with a longer tail. The test set performance improved to be as good as LambdaRank for the range $-0.4 < \alpha < 0.4$, reaching an optimum when $\alpha = 0.0$.

5. CONCLUSION

We have introduced the idea of using rank distributions to smooth traditional IR metrics. We have shown how to compute an approximation to these rank distributions in a way that involves no explicit sort, and is therefore differentiable with respect to the model parameters, and so suitable for memory efficient non-linear machine learning approaches such as the multi-layer perceptron. We have shown that the Rank-Binomial is a good and useful approximation to the true rank distributions by demonstrating that it can reliably find better training set NDCGs than state-of-the-art algorithms designed for that purpose.

We have shown also that SoftRank does not, in some cases, generalize to new queries as well as LambdaRank. We have shown that this is not due to lack of regularization, but rather a tendency to focus too much on the top ranks. We fixed this problem by trying a range of training discount functions that are less top-heavy, and found that it was possible to do as well as LambdaRank on the test set but, as yet, not significantly better.

We believe that SoftRank represents a general and powerful new approach for direct optimization of non-smooth ranking metrics. Future work will focus on characterizing better the conditions under which SoftNDCG fails to generalize, and exploring other soft IR metrics.

6. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incporporating user behavior information. In *SIGIR*, 2006.

[2] H. Balakrishnan, I. Hwang, and C. Tomlin. Polynomial approximation algorithms for belief matrix maintenance in identity management. In *IEEE Conference on Decision and Control*, 2004.

[3] C. Burges, R. Ragno, and Q. V. L. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, 2006.

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *ICML*, 2007.

[6] R. Carnuana, S. Baluja, and T. Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *NIPS 8*, 1996.

[7] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6:1019–1041, 2005.

[8] K. Crammer and Y. Singer. Pranking with ranking. In *NIPS 14*, 2002.

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

[10] Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[11] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of Knowledge Discovery in Databases*, 2002.

[12] Y. LeCun, L. Bottou, G. Orr, and K.-R. Müller. Efficient backprop, 1998.

[13] D. Metzler, T. Strohman, and W. Croft. Indri at trec 2006: Lessons learned from three terabyte tracks. In *online Proceedings of Text REtrieval Conference*, 2005.

[14] A. Papoulis. *Probability, Random Variables and Stochastic Processes, Third Edition*. McGraw-Hill 1991.

[15] S. Robertson, H. Zaragoza, and M. Taylor. A simple BM 25 extension to multiple weighted fields. In *CIKM*, pages 42–29, 2004.

[16] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM*, 2006.