Neural Learning-to-rank models: a comparison between objective functions Group 16

JIAYI TANG*, CĂTĂLIN LUPĂU, and MATTIA BONFANTI*, Delft University of Technology, The Netherlands

Learning-to-rank is a ranking task that predicts the relevance of instances using fixed-size input data with numerical features and ranks them using the output scores. Different approaches are available to perform Learning-to-rank such as pointwise, pairwise and listwise. The main challenge is presented by the fact that widely used loss functions cannot be used for this task since they are not well connected to the proper evaluation metrics. In the light of this, previous researches have been focusing on proposing suitable differentiable loss functions for Learning-to-rank. Additionally, the advancement in the field of neural networks has also been the subject of works that had the goal of applying this to improve Learning-to-rank. This paper aims to provide an empirical analysis of the performance of different loss functions (MseNet, RankNet, ApproxNDCG, NeuralNDCG) against different metrics (NDCG, NDCG@k, Precision@k, MAP), over the same neural network architecture. The methodology, the experiments setup and results are discussed to enhance reproducibility. To ensure the consistency and generalization of the results, the experiments are run on different datasets (*MQ2008* and *MLSR-WEB10K*). A discussion of the results together with limitations and future work reflections are then given. The work adds additional resources to the field of Learning-to-rank and can serve as a starting point and support material for further studies.

CCS Concepts: • **Information systems** → **Learning to rank**.

Additional Key Words and Phrases: learning to rank, objective functions, datasets, neural networks

ACM Reference Format:
Jiayi Tang, Cătălin Lupău, and Mattia Bonfanti. 2023. Neural Learning-to-rank models: a comparison between objective functions Group 16. In *IN4325-Q3-23: Information Retrieval, TU Deflt*. ACM, New York, NY, USA, 15 pages.

1 INTRODUCTION

Learning-to-rank is a ranking task that takes fixed-size data with numerical features as input, and predicts the relevance of each input instance. These relevance scores are then used to rank the items from highest to the lowest. The ranking models used to define the relevance of the items are usually implemented as either gradient boosted trees, multi-layer perceptrons and tranformers, and can be categorized as pointwise, pairwise or listwise. The pointwise approach frames the ranking task as a simple regression or classification of the true relevancy for input entities regardless of possible interactions between them. In the pairwise approach, pairs of input items are treated as independent variables and the loss functions is optimized around the preference among the pairs. Finally, the listwise approach computes the loss based on the scores of the entire input list [9].

However, previous research has shown how widely used loss functions are not well connected to the actual Information Retrieval (IR) metrics used for evaluating their performance. That being said, popular metrics like Normalised Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) present the challenge of not being differentiable, making

*All authors contributed equally to this research.

them not suitable for being loss functions. This challenge has been subject of different researches which contributed to the development of differentiable approximations of such metrics that can be used as loss functions in Learning-to-rank tasks [10]. On top of this, in recent years, the progress in the training of neural networks introduced an additional perspective of effectiveness and scalability of deep network architectures applied to Learning-to-rank [3]. Therefore, understanding the behavior and the performance of more suitable loss functions combined with more sophisticated neural models is a topic that should not be overlooked.

Another topic of concern for Learning-to-rank is the influence that the used datasets have on the quality of a model when inconsistency and bias are present. As the performance of supervised Machine Learning models is highly dependent on the quality of the training dataset, it is important that the used data presents consistency in the labels assigned to similar items. This translates to Learning-to-rank as well, where previous research has shown how inconsistency in the used datasets can negatively impact the performance of the model [5]. On top of this, the bias present in the datasets cannot be ignored either. As Unbiased Learning-to-rank techniques have been introduced to mitigate the position bias and the trust bias, some limitations are still present. For example, it has been shown how the bias related to other display information has been overlooked. This can be seen as a consequence to the fact that most existing datasets used for Learning-to-rank lack real-world user feedback [11]. Therefore, it is fundamental for this work to be aware of the potential bias and inconsistency a dataset can carry over, which can bring challenges when trying to generalize a model over different scenarios.

The aim of this paper is to present and discuss an empirical comparison of the performance of different loss functions against different ranking metrics, using the same underlying neural network architecture. On top of this, the experiments are run on different datasest to inspect if the eventual presence of bias and/or inconsistency can affect the loss functions performance. Thus, the following research questions will be answered:

(1) What is the best loss functions for Learning-to-rank applied to a neural network for each considered metric?
(2) Can the results of one dataset be generalized to other datasets?

Related work that this paper is built upon is presented in section 2. In section 3 the discussion on the used methodology includes: datasets; loss functions; metrics for comparisons. Section 4 illustrates the performed experiments with the setup, the results, the insights discussion and the limitations. The main takeaways and reflections on further work are discussed in the conclusion.

2 RELATED WORK

In their work from 2019, Bruch et al. discuss the usage of deep neural networks for directly optimizing the NDCG score. Their contribution focuses on the fine tuning of the hyperparameter alpha as well as investigating the impact of deeper networks and larger training datasets on the quality of the model. As Bruch et al. revisit the work from 2013 by Qin et al. [10], which proposed a framework to directly optimize ranking metrics, they apply the findings to ApproxNDCG (NDCG differentiable approximation). The study of the hyperparamters of this loss function serves as a foundation to illustrate the benefits that deep neural networks bring to Learning-to-rank [3]. The ApproxNDCG loss function will be used in this paper as one of the candidates for the performance analysis. The methodology used by Bruch et al. will be taken as inspiration and expanded to other comparison metrics rather than NDCG only.

An additional investigation of the performance of objective functions applied to neural network based Learning-to-rank is given by Burges et al. in their work from 2005. The investigation of using gradient descent in Learning-to-rank is supported by the introduction of RankNet, a neural network built on the underlying proposed ranking function [4].

As the results obtained by their experiments showed how well RankNet performs on real world ranking problems with large amounts of data [4], it will then be used in the empirical analysis of this paper. Unlike in the work by Burges et al., RankNet will be evaluated on different metrics rather than only NDCG and it will be compared with other loss functions applied to neural networks instead of just other linear systems.

Pobrotyn et al., in their work from 2021, address the issue of the mismatch between the optimisation objective and the evaluation criterion in the most common Learning-to-rank loss functions. They argue that this is due because some of the evaluation metrics used for Learning-to-rank algorithms have undefined derivatives, which makes them unsuitable for gradient-based optimisation. To overcome this they present NeuralNDCG, a differentiable approximation of the NDCG loss function [9]. The work provides extensive experiments and comparisons between different loss functions including ApproxNDCG, RankNet and LambdaRank. These loss functions will be compared in this paper to determine the most performant for different metrics, rather than NDCG only.

The work from 2021 by Faenza et al. discusses the importance of consistency in the datasets used for Learning-to-rank tasks by exploring the negative effect that inconsistency has on the models' performance. Their analysis shows how a Group Decision Making consensus process can be used to evaluate the consistency of the data and improve the performance of the used Learning-to-rank model [5]. The quality of datasets is also studied by Zou et al. in their work from 2022, where they inspect the problem of bias in the data and its effect on the final performance. In particular, they propose Baidu-ULTR, a new unbiased web search dataset that can be used for Learning-to-rank training [11]. The issues raised by these works about the quality of datasets will be considered throughout the execution of the current work. Therefore, the experiments will be run on different datasets to ensure the consistency of the results.

3 METHOD

The methodology used for the empirical evaluation is discussed in this section by presenting the used datasets (3.1), the chosen loss functions (3.2) and the chosen metrics (3.3).

3.1 Datasets

In their work from 2013, Qin et al. present several datasets that were used for research on Learning-to-rank. One of the datasets, called MQ2008 (3.1.1), is based on the million query track TREC2008 and contains about 800 queries with labeled documents. Another dataset, called MLSR-WEB (3.1.2), is based on queries gathered from Microsoft's Bing [10]. These two datasets have been used in this work to perform the empirical analysis of the performance of the selected loss functions over different metrics.

3.1.1 MQ2008. The MQ2008 dataset is part of the LETOR4.0 package presented by Qin et al. in their work from 2013. The LETOR4.0 package provides 4 different ranking settings for the MQ2008 dataset: supervised; semi-supervised; aggregation; listwise. The dataset of each ranking is divided in 5 folds with the split into train, test and validation sets already provided [10].

3.1.2 MLSR-WEB10K. The MLSR-WEB10K dataset consist of feature vectors derived from query-url pairs together with the relevance judgment labels. These labels are retrieved from a dismissed set generated by the Bing search engine and they can assume a value between 0 (irrelevant) and 4 (highly relevant). The features were extracted by Qin et al. as discussed in their work from 2013 [10]. Overall, MLSR-WEB10K contains about 10000 queries.

In the dataset, each row represents a 136-dimensional feature vector that refers to a query-url pair, and includes: the relevance judgement label; the query id; the remaining features. As for the MQ2008, 5 partitions are provided to perform cross-validation. All partitions include the split into train, test and validation sets [10].

3.2 Loss Functions

The algorithms for learning-to-rank can be grouped into three categorizations, namely the pointwise approach, the pairwise approach and the listwise approach.

3.2.1 The pointwise approach. The input of the pointwise approach always contains the feature vector of each single document and the output contains the relevance degree of each single document. In the pointwise approach, we use the neural network to predict the relevance degree of a document using the feature vector of the document as input.

The loss function of the pointwise approach examines the accuracy of the prediction for each single document. We simply have a square error as the loss function:

$$L(f)_{MseNet} = (y_j - f(x_j))^2 \tag{1}$$

where f is our neural network, y_j is the ground truth label and x_j is the feature vector of document j.

3.2.2 The pairwise approach. The pairwise approach does not focus on accurately predicting the relevance degree of each document like the pointwise approach. Instead, it cares about the relative order between two documents. The input of the pairwise approach contains a pair of documents both of which are represented as feature vectors and the output contains the pairwise preference between each pair of documents. When the relevance degree l_j of each document j is given, the pairwise preference for document pair (x_u, x_v) is defined as:

$$y_{u,v} = 2 \cdot I_{\{l_u > l_v\}} - 1 \tag{2}$$

where $I_{\{A\}}$ is an indicator function which is defined to be 1 if a predicate A holds and 0 otherwise.

In the pairwise approach, the loss function is always defined on a pair of documents and we have our experiments based on the loss function from RankNet [4]. Given two documents x_u and x_v associated with a training query q, a target probability $\bar{P}_{u,v}$ is constructed based on their ground truth labels. We define $\bar{P}_{u,v} = 1$ if $y_{u,v} = 1$ and $\bar{P}_{u,v} = 0$ otherwise. We use our neural network as a scoring function f and the modeled probability $P_{u,v}$ is defined based on the difference between the scores of these two documents:

$$P_{u,v}(f) = \frac{\exp(f(x_u) - f(x_v))}{1 + \exp(f(x_u) - f(x_v))} \tag{3}$$

where f is the scoring function (i.e., our neural network). x_u and x_v are the feature vectors of document u and v.

Then the cross entropy between the target probability and the modeled probability is used as the loss function:

$$L(f)_{RankNet} = -\bar{P}_{u,v} \log P_{u,v}(f) - (1 - \bar{P}_{u,v}) \log(1 - P_{u,v}(f)) \tag{4}$$

where $\bar{P}_{u,v}$ is defined by $y_{u,v}$ as mentioned and $P_{u,v}(f)$ is computed by Equation 3.

3.2.3 The listwise approach. One motivation for the listwise approach is to learn the ranking model by directly optimizing what is used to evaluate a ranking performance such as NDCG.

As compared to the pointwise and pairwise approaches, the advantage of list approaches lies in the fact that their loss functions can naturally consider the positions of documents in the ranked list. However, since these measures are position based, they are usually non-continuous and non-differentiable.

To tackle this challenge, one can choose to optimize a continuous and differentiable approximation of the IR evaluation measure. Following this direction, we have our experiments with the loss function from ApproxNDCG [3]. A smooth approximation of the rank of item i can be computed as:

$$\pi_f(i) = 1 + \sum_{j \neq i} \frac{1}{1 + e^{-\alpha(f(x_j) - f(x_i))}} \tag{5}$$

where $\alpha > 0$ is a knob that controls how tightly the sigmoid fits the indicator. Plugging this approximation into the ranking position in the formula of NDCG yields the loss function of ApproxNDCG:

$$L(f)_{ApproxNDCG} = N_n^{-1} \sum_{x \in \mathrm{X}} \frac{2^{r(x)} - 1}{\log_2(1 + \pi_f(x))} \tag{6}$$

where N_n^{-1} is the maxDCG given the ideal ranking, $r(x)$ is the relevance score of document x and $\pi_f(x)$ is computed using Equation 5.

One can also use a differentiable approximation of sorting like NeuralNDCG [9] to solve the problem. NeuralNDCG relies on NeuralSort [6], a smooth relaxation of the sorting operator.

Sorting a list of scores s is equivalent to left-multiplying a column vector of scores by the permutation matrix $P_{sort(s)}$ induced by permutation $sort(s)$ sorting the scores. Thus, in order to approximate the sorting operator, it is enough to approximate the induced permutation matrix. In NeuralSort, the permutation matrix is approximated via a unimodal row stochastic matrix $\widehat{P}_{sort(s)}$ given by:

$$\widehat{P}_{sort(s)}[i,:](\tau) = softmax[((n + 1 - 2i)s - A_s \mathbb{1})/\tau] \tag{7}$$

where A_s is the matrix of absolute pairwise differences of elements of s such that $A_s[i,j] = |s_i - s_j|$, $\mathbb{1}$ denotes the column vector of all ones and $\tau > 0$ is a temperature parameter controlling the accuracy of approximation.

Note that the temperature parameter τ allows to control the trade-off between the accuracy of the approximation and the variance of the gradients. Generally speaking, the lower the temperature, the better the approximation at the cost of a larger variance in the gradients:

$$\lim_{\tau \to 0} \widehat{P}_{sort(s)}(\tau) = P_{sort(s)} \tag{8}$$

Using the approximation of the sorting operator given by NeuralSort, NeuralNDCG first approximately sorts the relevancies using $\widehat{P}$ given in Equation 7 and multiplies the result by the logarithmic position discounts. The summation is done over the ranks of documents and the loss function of NeuralNDCG is computed as:

$$L(f)_{NeuralNDCG}(\tau) = N_n^{-1} \sum_{j=1}^{n} \frac{scale(\widehat{P}_f) \cdot (2^{r(x_j)} - 1)}{\log_2(1 + j)} \tag{9}$$

where N_n^{-1} is the maxDCG given the ideal ranking, $r(x_j)$ is the relevance score of document x_j and $\widehat{P}$ is computed based on Equation 7.

3.3 Metrics

To avoid biasing our results towards a single metric, three different and widely used metrics were chosen for the evaluation process, including Precision, Mean Average Precision (MAP) [1] and Normalized Discounted Cumulative Gain (NDCG) [7].

3.3.1 Precision. Precision at position k evaluates the relevance judgement of the top k positions of a ranked list using two levels (i.e., relevant and irrelevant), which is defined as:

$$Pre@k = \frac{1}{k}\sum_{j=1}^{k} r_j \tag{10}$$

where k denotes the truncation position and

$$r_j = \begin{cases} 1 & \text{if document at } j\text{-th position is relevant,} \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

3.3.2 Mean Average Precision. Mean Average Precision (MAP) is defined on the basis of Precision. First, we define the Average Precision (AP) as:

$$AP = \frac{1}{|N_+|}\sum_{j} r_j \times Pre@j \tag{12}$$

where $|N_+|$ is the total number of relevant documents with respect to the query. Given a ranked list for a query, we can compute an AP for this query. Then the mean value of AP over all the test queries is the MAP.

3.3.3 Normalized Discounted Cumulative Gain. While the aforementioned two metrics are mainly designed for two-level judgments (i.e., relevant and irrelevant), Normalized Discounted Cumulative Gain (NDCG) can leverage the relevance judgment in terms of multiple ordered levels, and has an explicit position discount factor in its definition.

NDCG is computed as:

$$NDCG = N_n^{-1}\sum_{j=1}^{n} g(r_j)d(j) \tag{13}$$

where $g(r_j)$ denotes the gain, which is defined as:

$$g(r_j) = 2^{r_j} - 1 \tag{14}$$

and $d(j)$ denotes the discount, which is defined as:

$$d(j) = \frac{1}{\log_2(1+j)} \tag{15}$$

r_j denotes the relevance level of the document ranked at j-th position. N_n is the maximum of $\sum_{j=1}^{n} g(r_j)d(j)$, induced by the ideal ranked list, where the documents are sorted by the ground truth relevance scores.

To evaluate the top k positions of a ranked list, NDCG@k is used in this paper as well and it is defined as:

$$NDCG@k = N_k^{-1}\sum_{j=1}^{k} g(r_j)d(j) \tag{16}$$

where k denotes the truncation position.

4 EXPERIMENTS

In order to answer the proposed research questions, a series of experiments was conducted in which a wide range of neural-network based ranking algorithms were benchmarked against 2 publicly available document-query datasets: *MQ2008* and *MLSR-WEB10K*.

The backbone architecture (layers configuration, activation functions) of each of the neural Learning-to-rank algorithms was kept the same across all of the experiments within one dataset. The reason for this was that the objective of our experiments consisted of comparing the behaviour of different loss functions, rather than optimizing the underlying neural network architecture

As described in the previous sections, we decided to test a wide range of Learning-to-rank algorithms, following a diverse set of approaches. As a result of this, the following algorithms were included in the benchmarks:

(1) **MseNet** - a pointwise ranking algorithm that uses a neural network to predict the rank of a document based on its features. As the name suggests, the loss function used to train the neural network is the mean squared error.
(2) **RankNet** - a pairwise ranking algorithm that sequentially propagates the features of a pair of documents through a neural network. The 2 outputs of the neural network are subtracted from each other and then fed into a sigmoid layer. The resulting value is intepreted as the probability that the first document will rank higher than the second document. The loss function used to train the learning algorithm is the Binary Cross Entropy Loss, as this function represents a reliable distance measure between 2 probabilities.
(3) **ApproxNDCG** - a listwise neural ranking algorithm that uses the ApproxNDCG loss function proposed by Pobrotyn et al. in 2020 [8]. All documents corresponding to a query are fed in parallel through the backbone neural network. The outputs are then fed through the proposed differentiable approximation of the NDCG metric, which acts as a loss for the learning model.
(4) **NeuralNDCG** - a listwise neural ranking algorithm that works similarly to the previous algorithm, but uses a different loss function. The loss function used to train the model is the NeuralNDCG proposed by Pobrotyn et al. in 2021 [9].

4.1 Setup

We created our own Pytorch-based implementations for each of the benchmarked learning algorithms. The reason for this is that we wanted to have control over how the underlying neural network is implemented in order to guarantee that all of the implementations use the same layer configurations and activation functions.

Before training the algorithms, a series of data pre-processing steps were conducted.

First and foremost, all datasets were split into a training set and an evaluation set on which the metrics were later computed.The dataset split was conducted in such a way that there were no query overlaps between the 2 sets. Around 10% of the queries ended up in the evaluation set. The rest of the queries were used for training.

In the case of the MQ2008 dataset, the entire evaluation set was used to compute the evaluation metrics. On the other hand, when evaluating the algorithms on the MLSR-WEB10K dataset, random samples of 100 queries taken from the evaluation set were used instead. The reason for this was to speed up the computations.

Since RankNet requires to be trained on pairs of documents, a transformation was applied on the training set. All documents corresponding to one query were paired in all possible combinations and a probability score was assigned to each of the pairs. A pair of documents would receive a probability score of 0.5 if their ranks were equal, a 1.0 if the first document ranked higher than the second document and a 0.0 otherwise.

Analyzing the data obtained by applying the above-mentioned transformation revealed that the vast majority of document pairs were assigned a probability score of 0.5, since the original dataset had many equal scores. Due to the fact that an imbalance dataset can impair the training process, we decided to apply upsampling on the dataset to increase the number of pairs with a probability score of 0.0 or 1.0.

Last but not least, all the datasets were normalized by subtracting the mean from each feature vector and dividing it by its standard deviation.

4.2 Training

As mentioned previously, the algorithms were trained on two distinct datasets: *MQ2008* and *MLSR-WEB10K*. In order to perform hyper-parameter optimization, the training set obtained during the setup phase was split into a smaller training set (containing 70% of the data from the original training set) and a test set (containing 30% of the data).

Two different training strategies were used for some of the algorithms, as indicated by the suffix added to their names (i.e. QbQ - query by query training Batched - batched training). Query by query training refers to training the neural network in batches that correspond to one query. This means that every batch of documents is formed by taking the documents associated to one particular query. In contrast, in batched training, queries that have the same number of associated documents are grouped together. Thus, one training batch for the neural network ends up consisting of the union of documents corresponding to all the queries in one query group. The main advantage of batched training with respect to query by query training is that we paralelize more computations. However, this comes at the expense of more GPU memory being used.

The following sections provide a brief overview of the hyperparameters used for each algorithm and dataset, as well as diagrams showcasing the learning curves obtained during the training process.

4.2.1 MQ2008. Figure 1a showcases the train and test losses obtained while training MseNet. The network was trained using the Adam optimizer[2] with a learning rate of 0.001 and a batch size of 64. The network is made up of an input layer of 46 neurons connected to a 5 neurons hidden layer and an output layer of only one neuron.

In contrast, in figure 1b the learning curve of RankNet can be observed. The network was trained using the same parameters as MseNet, with the exception of the learning rate, which was 0.0001.

Both diagrams indicate that the learning process is smooth and no significant overfitting occurred.

Figure 2 shows the train and test losses obtained while training ApproxNDCG. The same network was used and trained using the same optimizer as the MseNet and RankNet, however, with a different learning rate of 0.0001 and a batch size of 8. Figure 3 shows the same setting of ApproxNDCG trained on the same dataset, with the exception of the batch size, which was 1, meaning that ApproxNDCG was traiend query by query.

The two diagrams suggest that when training with batches, ApproxNDCG has the problem of overfitting and the training process is not working as expected, since the test loss is much higher than the train loss and losses are not decreasing much as the number of epoch increases. However, when training with batch size of 1 (i.e., query by query), the problem of overfitting has been alleviated a lot and the learning process was much more efficient.

Figure 4 showcases the train and test losses while training NeuralNDCG with batches. The setting was the same as ApproxNDCG, with the same network, the Adam optimizer, a learning rate of 0.0001 and a batch size of 8. Unlike ApproxNDCG, the trainig process of NeuralNDCG is more smooth since the losses decreases as the number of epoch increases. However, the large gap between the train and test loss indicates that the problem of overfitting still exists.

4.2.2 MLSR-WEB10K. Figure 5 and Figure 6 shows the train and test losses obtained when training MseNet on dataset *MLSR-WEB10K*. The same network was used as on dataset *MQ2008*. The Adam optimizer was used and the learning rate was 0.001. When training with batches, the batch size was 64 and when training query by query, the batch size was 1. The two diagrams indicate that training with query by query helps to alleviate the problem of overfitting and improve the training process.

Figure 7 and Figure 8 shows the train and test losses obtained when training ApproxNDCG. The same network and optimizer were used. However, the training rate was 0.0001 and when training with batches, the batch size was 8. When training query by query, the batch size was still 1. A similar suggestion of less overfitting and a better training process can also be reached from the two diagrams.

Figure 9 showcases the train and test losses while training NeuralNDCG. The same network, optimizer, training rate, and batch size as those in ApproxNDCG were used. Compared to ApporxNDCG which is also a listwise approach, the learning curves of NeuralNDCG trained with batches are much more smooth and the training process is much more efficient. However, since the gap between the train and test loss increases as the number of epoch increases, the problem of overfitting begins to appear but it is not that severe.

4.3 Results Discussion

Dataset	**Algorithm**	**NDCG**	**NDCG@2**	**NDCG@4**	**NDCG@6**	**Pre@2**	**Pre@4**	**Pre@6**	**MAP**
MQ2008	MseNetBatched	0.79	0.47	0.56	0.63	0.28	0.26	0.27	0.45
MQ2008	RankNet	0.85	0.61	0.68	0.74	0.47	0.41	0.37	0.56
MQ2008	ApproxNDCGBached	0.82	0.55	0.63	0.67	0.39	0.36	0.31	0.49
MQ2008	ApproxNDCGQbQ	0.87	0.66	0.72	0.76	0.53	0.46	0.39	0.63
MQ2008	NeuralNDCGBatched	**0.89**	**0.74**	0.79	**0.88**	0.60	0.47	**0.44**	0.74
MQ2008	NeuralNDCGQbQ	**0.89**	**0.74**	**0.8**	0.86	**0.62**	**0.52**	0.40	**0.75**
MLSR10K	MseNetQbQ	0.23	0.14	0.1	0.086	0.44	0.44	0.44	0.44
MLSR10K	MseNetBatched	0.67	0.16	0.18	0.20	0.26	0.302	0.308	0.43
MLSR10K	RankNet	-	-	-	-	-	-	-	-
MLSR10K	ApproxNDCGBatched	**0.79**	0.43	**0.47**	0.48	**0.78**	**0.76**	**0.72**	0.61
MLSR10K	ApproxNDCGQbQ	**0.79**	0.45	**0.47**	**0.49**	0.77	0.73	0.71	**0.63**
MLSR10K	NeuralNDCGBatched	**0.79**	0.**46**	0.45	0.46	0.70	0.686	0.654	0.611
MLSR10K	NeuralNDCGQbQ	-	-	-	-	-	-	-	-

Table 1. Metrics Results for Each Dataset/Algorithm

Table 1 contains the results that we obtained by running the experiments. What can be noticed is that for both datasets, the listwise neural ranking algorithms consistently outperform the pairwise and pointwise algorithms. The superior performance of the listwise methods on NDCG based metrics can be partly attributed to bias. Since the listwise methods were trained using loss functions that approximate NDCG, they are clearly more prone towards performing better according to NDCG metrics.

What is interesting, however, is the fact that the listwise algorithms outperform other categories of algorithms on precision-based metrics as well. This indicates that the listwise methods represent a superior class of neural ranking algorithms. While the exact reason for why this is the case would require further investigation, we can give the reader our speculation.

From a theoretical perspective, no machine learning algorithm is, in absolute terms, better than any other. Having said this, certain learning algorithms require less data to achieve a good performance at a certain task than others. As an example, both a fully connected neural network and a convolutional neural network can learn to classify images, but getting a convolutional neural network to achieve a good performance at image classification requires much less data. The reason why we observe this phenomenon is due to a property called prior. A machine learning model has a good prior when, even before starting training, the model approximates a probability distribution that is close to the real distribution of the dataset.

Our supposition for why listwise methods achieve better performance is that they have a better prior with respect to the document ranking problem than the other models. The MseNet model is very generalistic and could be applied to any regression problem. The RankNet model is a bit more specific, since it learns to compare 2 documents and predict which should come first, which is closer to the idea of ranking. The listwise methods, however, are highly coupled with the problem of ranking, as the output of a listwise model is a full ordering of the documents associated to a query. Hence, listwise methods sacrifice generality by introducing information about the problem into the model in order to increase the prior.

Coming back to the analysis of the experimental results, our data seems to be inconclusive when it comes to comparing ApproxNDCG with NeuralNDCG. Based on our experiments alone, it cannot be said with certainty which of the 2 is better. It should be noted, however, that training ApproxNDCG requires significant less time than training NeuralNDCG.

4.4 Limitations

While performing the experiments we faced some limitations due to lack of computational resources. More specifically, we were not able to train RankNet and NeuralNDCGQbQ (NeuralNDCG trained query by query) on the larger MLSR10K dataset.

The reason why RankNet could not be trained was that the data would have required pre-processing. The script we used to pre-process the MQ2008 dataset for Ranknet proved to be too slow for the MLSR10K dataset. We also tried a GPU accelerated algorithm, however, we ran out of GPU memory due to the dataset being too large. We believe that a potential solution to this problem could be to only apply the pre-processing step on the batches used during training, while making sure the data is loaded from primary storage to memory sequentially, rather than all at once. This technical problem could be tackled in a future iteration of the project, as our strict deadline did not allow us to tackle it in the current one.

The reason why NeuralNDCGQbQ (NeuralNDCG trained query by query) could not be trained is that the loss function of this model is very computationally expensive. Training this model on a large dataset and without too much parallelization (query by query) proved infeasible given our computational resources. Luckily, we were able to train the batched version of NeuralNDCG which benefits from more parallelization.

4.5 Reproducibility

The discussed experiments and their results can be reproduced and expanded using the code at the following repository:

https://github.com/catalinlup/learning-to-rank

5 CONCLUSION

The aim of this paper was to present an empirical evaluation of the performance of different loss functions against different ranking metrics, using the same neural network architecture. A check for the consistency and the generalization of the results over different datasets was also conducted. An analysis of related work on the subject was performed to gain a better perspective on the current approaches used for Learning-to-rank tasks. This allowed the gathering of the loss functions (MseNet, RankNet, ApproxNDCG, NeuralNDCG) and the ranking metrics (NDCG, NDCG@k, Precision@k, MAP) that were then used in the evaluation. On top of this, previous researches about data consistency and bias supported the decision to run the experiments on different datasets: *MQ2008* and *MLSR-WEB10K*. A simple neural network architecture has been developed for the training and evaluation of the loss functions.

According to our experimental results, the listwise ranking algorithms seem to outperform the other types of algorithms on both metrics. This result is consistent across most metrics and both datasets, which answers both of our research questions.

Overall, this paper fulfilled the aim of providing an empirical evaluation of the performance of different loss functions against different ranking metrics, over the same neural network architecture. The aim of checking the generalization and the consistency of the results over different datasets was also fulfilled. The presented methodology and results can serve as valuable resources that future work on the topic can build upon.

6 SELF-ASSESSMENT OF CONTRIBUTION

6.1 Jiayi Tang

The responsibilities over the project include gathering related papers, inspection of current methodologies, contributing to the definition of the method and experiments, supporting the implementation of ApproxNDCG and metrics, writing the Method and part of the Training section of the paper and proof reading.

6.2 Cătălin Lupău

My responsibilities for this project consisted of creating the Pytorch implementation of the models, setting up the experiments, running the experiments, as well interpreting the experimental results. I also wrote the discussion of the results and the limitations sections of the report.

6.3 Mattia Bonfanti

The responsibilities over the project were related to: gathering related papers; inspection of current methodologies; contributing to the definition of the method and experiments; supporting the implementation of the models; writing several sections of the paper (abstract, introduction, related work, datasets, conclusion); proof reading and fixing of grammar.

REFERENCES

[1] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.

[2] Sebastian Bock and Martin Weiß. 2019. A Proof of Local Convergence for the Adam Optimizer. In *2019 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN.2019.8852239

[3] Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and Marc Najork. 2019. Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks. 1241–1244. https://doi.org/10.1145/3331184.3331347

[4] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory Hullender. 2005. Learning to Rank using Gradient Descent. *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, 89–96. https://doi.org/10.1145/1102351.1102363

[5] Giuseppe Fenza, Mariacristina Gallo, Vincenzo Loia, Francesco Orciuoli, and Enrique Herrera-Viedma. 2021. Data set quality in Machine Learning: Consistency measure based on Group Decision Making. *Applied Soft Computing* 106 (2021), 107366. https://doi.org/10.1016/j.asoc.2021.107366

[6] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850* (2019).

[7] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[8] Przemyslaw Pobrotyn, Tomasz Bartczak, Mikolaj Synowiec, Radoslaw Bialobrzeski, and Jaroslaw Bojar. 2020. Context-Aware Learning to Rank with Self-Attention. *ArXiv* abs/2005.10084 (2020).

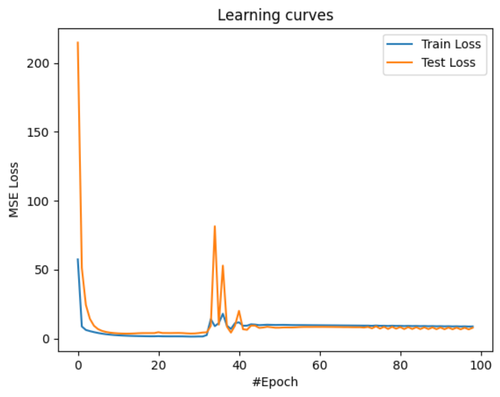
[9] Przemyslaw Pobrotyn and Radoslaw Bialobrzeski. 2021. NeuralNDCG: Direct Optimisation of a Ranking Metric via Differentiable Relaxation of Sorting. *ArXiv* abs/2102.07831 (2021).

[10] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

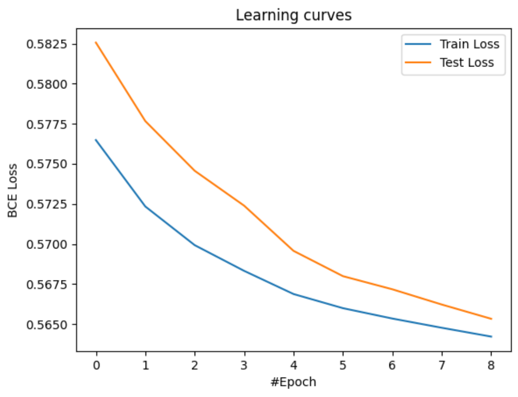
[11] Lixin Zou, Haitao Mao, Xiaokai Chu, Jiliang Tang, Wenwen Ye, Shuaiqiang Wang, and Dawei Yin. 2022. A Large Scale Search Dataset for Unbiased Learning to Rank. arXiv:2207.03051 [cs.AI]

A LEARNING CURVES ON MQ2008

The learning curves of MseNet, RankNet, ApproxNDCG and NeuralNDCG trained on dataset *MQ2008* are provided below.



(a) Learning curves obtained while training MseNet.



(b) Learning curves obtained while training RankNet.

Fig. 1. Learning curves while training on the MQ2008 dataset.

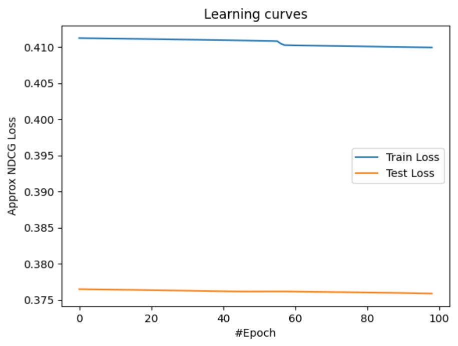


Fig. 2. Learning curves of ApproxNDCG trained with batches

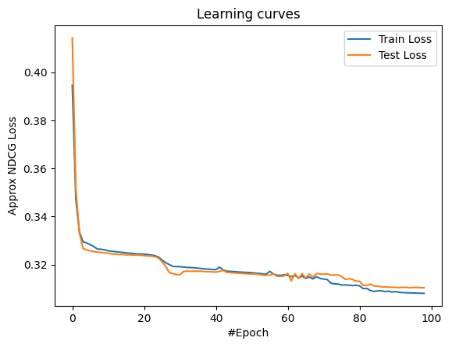


Fig. 3. Learning curves of ApproxNDCG trained query by query

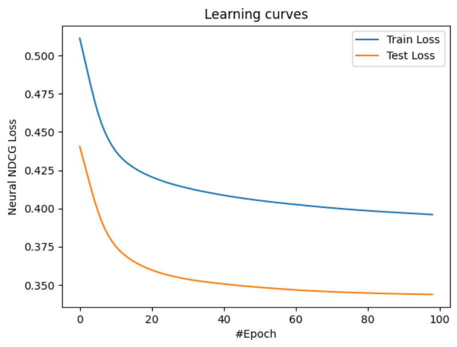


Fig. 4. Learning curves of NeuralNDCG trained with batches

B LEARNING CURVES ON MLSR-WEB10K

The learning curves of MseNet, RankNet, ApproxNDCG and NeuralNDCG trained on dataset *MLSR-WEB10K* are provided below.

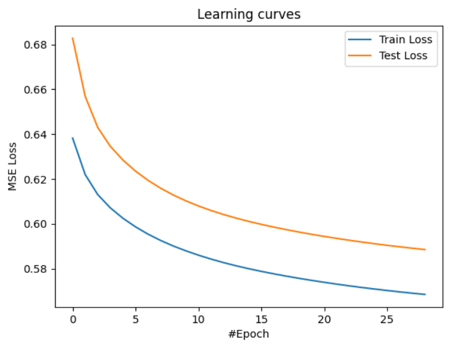


Fig. 5. Learning curves of MseNet trained with batches

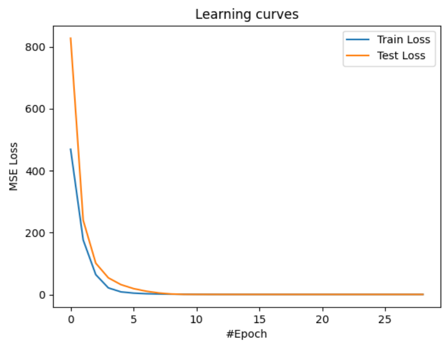


Fig. 6. Learning curves of MseNet trained query by query

Received 07 April 2023; revised 07 April 2023; accepted 07 April 2023

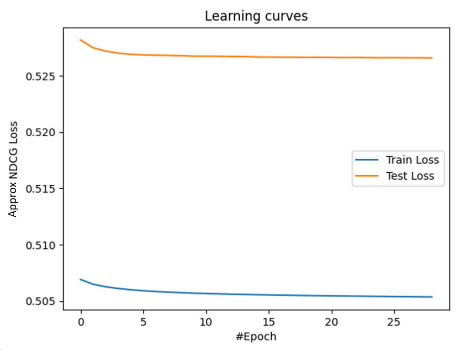


Fig. 7. Learning curves of ApproxNDCG trained with batches

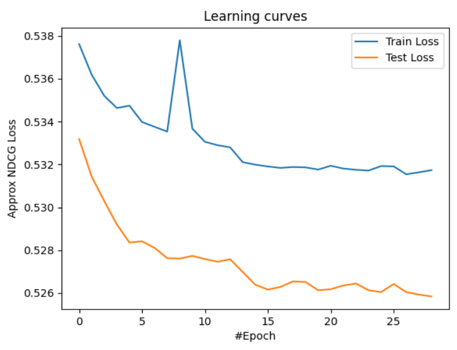


Fig. 8. Learning curves of ApproxNDCG trained query by query

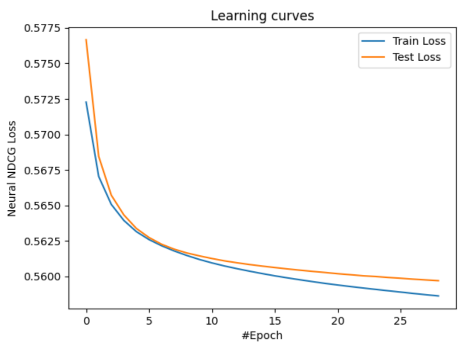


Fig. 9. Learning curves of NeuralNDCG trained with batches