# Context-Aware Learning to Rank with Self-Attention

Przemysław Pobrotyn
ML Research at Allegro.pl
przemyslaw.pobrotyn@allegro.pl

Tomasz Bartczak
ML Research at Allegro.pl
tomasz.bartczak@allegro.pl

Mikołaj Synowiec
ML Research at Allegro.pl
mikolaj.synowiec@allegro.pl

Radosław Białobrzeski
ML Research at Allegro.pl
radoslaw.bialobrzeski@allegro.pl

Jarosław Bojar
ML Research at Allegro.pl
jaroslaw.bojar@allegro.pl

## ABSTRACT

In learning to rank, one is interested in optimising the global ordering of a list of items according to their utility for users. Popular approaches learn a scoring function that scores items individually (i.e. without the context of other items in the list) by optimising a pointwise, pairwise or listwise loss. The list is then sorted in the descending order of the scores. Possible interactions between items present in the same list are taken into account in the training phase at the loss level. However, during inference, items are scored individually, and possible interactions between them are not considered. In this paper, we propose a context-aware neural network model that learns item scores by applying a self-attention mechanism. The relevance of a given item is thus determined in the context of all other items present in the list, both in training and in inference. We empirically demonstrate significant performance gains of self-attention based neural architecture over Multi-Layer Perceptron baselines. This effect is consistent across popular pointwise, pairwise and listwise losses on datasets with both implicit and explicit relevance feedback. Finally, we report new state-of-the-art results on MSLR-WEB30K, the learning to rank benchmark.

## CCS CONCEPTS

• **Information systems** → **Learning to rank**; *Information retrieval*;

## KEYWORDS

learning to rank, self-attention, context-aware ranking

**ACM Reference Format:**
Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. 2010. Context-Aware Learning to Rank with Self-Attention. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Learning to rank (LTR) is an important area of machine learning research, lying at the core of many information retrieval (IR) systems. It arises in numerous industrial applications like search engines, recommender systems, question-answering systems, and others.

A typical machine learning solution to the LTR problem involves learning a scoring function, which assigns real-valued scores to each item of a given list, based on a dataset of item features and human-curated or implicit (e.g. clickthrough logs) relevance labels. Items are then sorted in the descending order of scores [20]. Performance of the trained scoring function is usually evaluated using an IR metric like Mean Reciprocal Rank (MRR) [30], Normalised Discounted Cumulative Gain (NDCG) [16] or Mean Average Precision (MAP) [4].

In contrast to other classic machine learning problems like classification or regression, the main goal of a ranking algorithm is to determine relative preference among a group of items. Scoring items individually is a proxy of the actual learning to rank task. Users' preference for a given item on a list depends on other items present in the same list: an otherwise preferable item might become less relevant in the presence of other, more relevant items. Common learning to rank algorithms attempt to model such inter-item dependencies at the loss level. That is, items in a list are still scored individually, but the effect of their interactions on evaluation metrics is accounted for in the loss function, which usually takes a form of a pairwise (RankNet [6], LambdaLoss [31]) or a listwise (ListNet [9], ListMLE [32]) objective. For example, in LambdaMART [8] the gradient of the pairwise loss is rescaled by the change in NDCG of the list which would occur if a pair of items was swapped. Pointwise objectives, on the other hand, do not take such dependencies into account.

In this work, we propose a learnable, context-aware, self-attention [28] based scoring function, which allows for modelling of inter-item dependencies not only at the loss level but also in the computation of items' scores. Self-attention is a mechanism first introduced in the context of natural language processing. Unlike RNNs [14], it does not process the input items sequentially but allows the model to attend to different parts of the input regardless of their distance from the currently processed item. We adapt the Transformer [28], a popular self-attention based neural machine translation architecture, to the ranking task. Since the self-attention operation is permutation-equivariant (scores items the same way irrespective of their input order), we obtain a permutation-equivariant scoring function suitable for ranking. If we further refine the model with positional encodings, the obtained model becomes suitable for re-ranking setting. We demonstrate that the obtained (re)ranking model significantly improves performance over Multi-Layer Perceptron (MLP) baselines across a range of pointwise, pairwise and listwise ranking losses. Evaluation is conducted on MSLR-WEB30K [25], the benchmark LTR dataset with multi-level relevance judgements, as well as on clickthrough data coming from Allegro.pl, a


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

large-scale e-commerce search engine. We also establish the new state-of-the-art results on WEB30K in terms of NDCG@5.

We provide an open-source Pytorch [23] implementation of our self-attentive context-aware ranker available at https://github.com/allegro/allRank.

The rest of the paper is organised as follows. In Section 2 we review related work. In Section 3 we formulate the problem solved in this work. In Section 4 we describe our self-attentive ranking model. Experimental results and their discussion are presented in Section 5. In Section 6 we conduct an ablation study of various hyperparameters of our model. Finally, a summary of our work is given in Section 7.

## 2 RELATED WORK

Learning to rank has been extensively studied and there is a plethora of resources available on classic pointwise, pairwise and listwise approaches. We refer the reader to [20] for the overview of the most popular methods.

What the majority of LTR methods have in common is that their scoring functions score items individually. Inter-item dependencies are (if at all) taken into account at the loss level only. Previous attempts at modelling context of other items in a list in the scoring function include:

- a pairwise scoring function [12] and Groupwise Scoring Function (GSF) [2], which incorporates the former work as its special case. However, the proposed GSF method simply concatenates feature vectors of multiple items and passes them through an MLP. To desensitize the model to the order of concatenated items, Monte-Carlo sampling is used, which yields an unscalable algorithm,
- a seq2slate model [5] uses an RNN combined with a variant of Pointer Networks [29] in an encoder-decoder type architecture to both encode items in a context-aware fashion and then produce the optimal list by selecting items one-by-one. Authors evaluate their approach only on clickthrough data (both real and simulated from WEB30K). A similar, simpler approach known as Deep Listwise Context Model (DLCM) was proposed in [1]: an RNN is used to encode a set of items for re-ranking, followed by a single decoding step with attention,
- in [15], authors attempt to capture inter-item dependencies by adding so-called *delta* features that represent how different a given item is from items surrounding it in the list. It can be seen as a simplified version of a local self-attention mechanism. Authors evaluate their approach on proprietary search logs only,
- authors of [17] formulate the problem of re-ranking of a list of items as that of a whole-list generation. They introduce ListCVAE, a variant of Conditional Variational Auto-Encoder [26] which learns the joint distribution of items in a list conditioned on users' relevance feedback and uses it to directly generate a ranked list of items. Authors claim NDCG unfairly favours greedy ranking methods and thus do not use that metric in their evaluation,
- similarly to our approach, Pei et al. [24] use the self-attention mechanism to model inter-item dependencies. Their approach, however, was not evaluated on a standard WEB30K dataset and the only loss function considered was ListNet.

Our proposed solution to the problem of context-aware ranking makes use of the self-attention mechanism. It was first introduced as intra-attention in [11] and received more attention after the introduction of the Transformer architecture [28]. Our model can be seen as a special case of the encoder part of the Transformer.

Our model, being a neural network, can be trained with gradient-based optimisation methods to minimise any differentiable loss function. Loss functions suitable for the ranking setting have been studied extensively in the past [20]. In order to demonstrate that our context-aware model provides performance boosts irrespectively of the loss function used, we evaluate its performance when tasked with optimising several popular ranking losses.

We compare the proposed approach with those of the aforementioned methods which provided an evaluation on WEB30K in terms of NDCG@5. These include GSF of [2] and DLCM of [1]. We outperform both competing methods.

## 3 PROBLEM FORMULATION

In this section, we formulate problem at hand in learning to rank setting. Let $X$ be the training set. It consists of pairs $(\boldsymbol{x}, \boldsymbol{y})$ of a list $\boldsymbol{x}$ of $d_f$-dimensional real-valued vectors $x_i$ together with a list $\boldsymbol{y}$ of their relevance labels $y_i$ (multi-level or binary). Note that lists $\boldsymbol{x}$ in the training set may be of varying length. The goal is to find a scoring function $f$ which maximises an IR metric of choice (e.g. NDCG) on the test set. Since IR metrics are rank based (thus, non-differentiable), the scoring function $f$ is trained to minimise the average of a surrogate loss $l$ over the training data.

$$\mathcal{L}(f) = \frac{1}{|X|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in X} l((\boldsymbol{x}, \boldsymbol{y}), f),$$

while controlling for overfitting (e.g. by using dropout [27] in the neural network based scoring function $f$ or adding $L_1$ or $L_2$ penalty term [21] to the loss function $l$ ). Thus, two crucial choices one needs to make when proposing a learning to rank algorithm are that of a scoring function $f$ and loss function $l$. As discussed earlier, typically, $f$ scores elements $x_i \in \boldsymbol{x}$ individually to produce scores $f(x_i)$, which are then input to loss function $l$ together with ground truth labels $y_i$. In subsequent sections, we describe our construction of context-aware scoring function $f$ which can model interactions between items $x_i$ in a list $\boldsymbol{x}$. Our model is generic enough to be applicable with any of standard pointwise, pairwise or listwise loss. We thus experiment with a variety of popular ranking losses $l$.

## 4 SELF-ATTENTIVE RANKER

In this section, we describe the architecture of our self-attention based ranking model. We modify the Transformer architecture to work in the ranking setting and obtain a scoring function which, when scoring a single item, takes into account all other items present in the same list.

## 4.1 Self-Attention Mechanism

The key component of our model is the self-attention mechanism introduced in [28]. The attention mechanism can be described as taking the query vector and pairs of key and value vectors as input and producing a vector output. The output of the attention mechanism for a given query is a weighted sum of the value vectors, where weights represent how relevant to the query is the key of the corresponding value vector. Self-attention is a variant of attention in which query, key and value vectors are all the same - in our case, they are vector representations of items in the list. The goal of the self-attention mechanism is to compute a new, higher-level representation for each item in a list, by taking a weighted sum over all items in a list according to weights representing the relevance of these items to the query item.

There are many ways in which one may compute the relevance of key vectors to query vectors. We use the variant of self-attention known as *Scaled Dot-Product Attention*. Suppose $Q$ is a $d_{\text{model}}$-dimensional matrix representing all items (queries) in the list. Let $K$ and $V$ be the keys and values matrices, respectively. Then

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_{\text{model}}}})V.$$

The scaling factor of $\frac{1}{\sqrt{d_{\text{model}}}}$ is added to avoid small gradients in the softmax operation for large values of $d_{\text{model}}$.

## 4.2 Multi-Headed Self-Attention

As described in [28], it is beneficial to perform the self-attention operation multiple times and concatenate the outputs. To avoid growing the size of the resulting output vector, matrices $Q$, $K$ and $V$ are first linearly projected $H$ times to $d_q$, $d_k$ and $d_v$ dimensional spaces, respectively. Usually, $d_q = d_k = d_v = d_{\text{model}}/H$. Each of $H$ computations of a linear projection of $Q$, $K$, $V$, followed by a self-attention mechanism is referred to as a single attention head. Note that each head has its own learnable projection matrices. The outputs of each head are concatenated and once again linearly projected, usually to the vector space of the same dimension as that of input matrix $Q$. Similarly to the Transformer, our model also uses multiple attention heads. Thus

$$\text{MultiHead(Q, K, V)} = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{H})W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

and the projections are given by matrices

$$W_i^Q \in \mathbb{R}^{d_{model} \times d_q},$$

$$W_i^K \in \mathbb{R}^{d_{model} \times d_k},$$

$$W_i^V \in \mathbb{R}^{d_{model} \times d_v},$$

$$W^O \in \mathbb{R}^{Hd_v \times d_{model}}.$$

## 4.3 Permutation-equivariance

The key property of the proposed context-aware model making it suitable for the ranking setting is that it is permutation-equivariant, i.e. the scores of items do not depend on the original ordering of the input. Recall the definition of permutation equivariance:

*Definition 4.1.* Let $\boldsymbol{x} \in \mathbb{R}^n$ be a real-valued vector and $\pi \in S_n$ be a permutation of $n$ elements. A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is called permutation-equivariant iff

$$f(\pi(\boldsymbol{x})) = \pi(f(\boldsymbol{x})).$$

That is, a function is permutation-equivariant if it commutes with any permutation of the input elements.

It is a trivial observation that the self-attention operation is permutation-equivariant.

## 4.4 Positional Encodings

Transformer architecture was designed to solve a neural machine translation (NMT) task. In NMT, the order of input tokens should be taken into account. Unlike RNNs, self-attention based encoder has no way of discerning the order of input tokens, because of its permutation-equivariance. Authors of the original Transformer paper proposed to solve the problem by adding either fixed or learnable positional encodings to the input embeddings. Fixed positional encodings use sine and cosine functions of different frequencies, as follows:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

where *pos* is the position and *i* is the dimension.

The ranking problem can be viewed as either ordering a set of (unordered) items or as re-ranking, where the input list has already been sorted according to a weak ranking model. In the former case, the use of positional encodings is not needed. In the latter, they may boost the model's performance. We experiment with both ranking and re-ranking settings and when positional encodings are used, we test the fixed encodings variant [1]. Details can be found in Section 5.

## 4.5 Model Architecture

We adapt the Transformer model to the ranking setting as follows. Items on a list are treated as tokens and item features as input token embeddings. We denote the length of an input list as $l$ and the number of features as $d_f$. Each item is first passed through a shared fully connected layer of size $d_{fc}$. Next, hidden representations are passed through an encoder part of Transformer architecture with $N$ encoder blocks, $H$ heads and hidden dimension $d_h$. Recall that an encoder block in the Transformer consists of a multi-head attention layer with a skip-connection [13] to the input, followed by layer normalisation [3], time-distributed feed-forward layer, and another skip connection followed by layer normalisation. Dropout is applied before performing summation in residual blocks. Finally, after $N$ encoder blocks, a fully-connected layer shared across all items in the list is used to compute a score for each item. The model can be seen as an encoder part of the Transformer with extra linear projection on the input (see Figure 1 for a schematic of the architecture). By using self-attention in the encoder, we ensure that in the computation of a score of a given item, hidden representation of all other items were accounted for. Obtained scores, together with ground truth labels, can provide input to any ranking loss of choice. If the loss is a differentiable function of scores (and thus, of model's parameters),

[1] We found learnable positional encodings to yield similar results.

one can use SGD to optimise it. We thus obtain a general, context-aware model for scoring items on a list that can readily be used with any differentiable ranking loss. Since all the components used in the construction of the model (self-attention, layer normalisation, feed-forward layers) are permutation-equivariant, the entire model is permutation-equivariant (unless positional encodings are used).

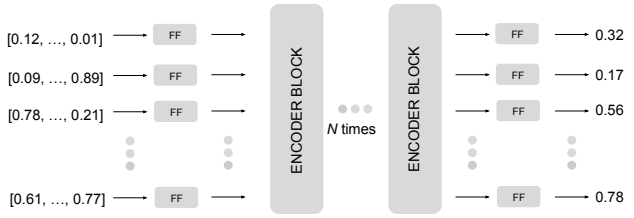


**Figure 1: Schematic of the proposed model architecture. Input is a list of real-valued vectors. Output is the list of real-valued scores.**

# 5 EXPERIMENTS

## 5.1 Datasets

Learning to rank datasets come in two flavours: they can have either multi-level or binary relevance labels. Usually, multi-level relevance labels are human-curated, whereas binary labels are derived from clickthrough logs and are considered implicit feedback. We evaluate our context-aware ranker on both types of data.

For the first type, we use the popular WEB30K dataset, which consists of more than 30,000 queries together with lists of associated search results. Every search result is encoded as a 136-dimensional real-valued vector and has associated with it a relevance label on the scale from 0 (irrelevant) to 4 (most relevant). We standardise the features before inputting them into a learning algorithm. The dataset comes partitioned into five folds with roughly the same number of queries per fold. We perform 5-fold cross-validation by training our models on three folds, validating on one and testing on the final fold. All results reported are averages across five folds together with the standard deviation of results. Since lists in the dataset are of unequal length, we pad or subsample to equal length for training, but use full length (i.e. pad to maximum length present in the dataset) for validation and testing. Note that there are 982 queries for which the associated search results list contains no relevant documents (i.e. all documents have label 0). For such lists, the NDCG can be arbitrarily set to either 0 or 1. To allow for a fair comparison with the current state-of-the-art, we followed LigthGBM [18] implementation of setting NDCG of such lists to 1 during evaluation.

For a dataset with binary labels, we use clickthrough logs of a large scale e-commerce search engine from Allegro.pl. The search engine already has a ranking model deployed, which is trained using XGBoost [10] with `rank:pairwise` loss. We thus treat learning on this dataset as a re-ranking problem and use fixed positional encodings in context-aware scoring functions. This lets the models leverage items' positions returned by the base ranker. The search logs consist of 1M lists, each of length at most 60. Nearly all lists (95%) have only one relevant item with label 1; remaining items were not clicked and are deemed irrelevant (label 0). Each item in a list is represented by a 45-dimensional, real-valued vector. We do not perform cross-validation on this set, but we use the usual train, validation and test splits of the data.

## 5.2 Loss Functions

To evaluate the performance of the proposed context-aware ranking model, we use several popular ranking losses. Pointwise losses used are RMSE of predicted scores and ordinal loss [22] (with minor modification to make it suitable for ranking). For pairwise losses, we use NDCGLoss 2++ (one of the losses of LambdaLoss framework) and its special cases, RankNet and LambdaRank [7]. Listwise losses used consist of ListNet and ListMLE.

Below, we briefly describe all of the losses used. For a more thorough treatment, please refer to the original papers. Throughout, $X$ denotes the training set, $\boldsymbol{x}$ denotes an input list of items, $\boldsymbol{s} = f(\boldsymbol{x})$ is a vector of scores obtained via the ranking function $f$ and $\boldsymbol{y}$ is the vector of ground truth relevancy labels.

*5.2.1 Pointwise RMSE.* The simplest baseline is a pointwise loss, in which no interaction between items is taken into account. We use RMSE loss:

$$l(\boldsymbol{s}, \boldsymbol{y}) = \sqrt{\sum_i (y_i - s_i)^2}$$

In practice, we used sigmoid activation function on the outputs of the scoring function $f$ and rescaled them by multiplying by maximum relevance value (e.g. 4 for WEB30K).

*5.2.2 Ordinal Loss.* We formulated ordinal loss as follows. Multi-level ground truth labels were converted to vectors as follows:

$$\begin{aligned} 0 &\mapsto [0, 0, 0, 0], \\ 1 &\mapsto [1, 0, 0, 0], \\ 2 &\mapsto [1, 1, 0, 0], \\ 3 &\mapsto [1, 1, 1, 0], \\ 4 &\mapsto [1, 1, 1, 1]. \end{aligned}$$

The self-attentive scoring function was modified to return four outputs and each output was passed through a sigmoid activation function. Thus, each neuron of the output predicts a single relevancy level, but by the reformulation of ground truth, their relative order is maintained, i.e. if, say, label 2 is predicted, label 1 should be predicted as well (although it is not strictly enforced and model is allowed to predict label 2 without predicting label 1). The final loss value is the mean of binary cross-entropy losses for each relevancy level. During inference, the outputs of all output neurons are summed to produce the final score of an item. Note that this is the classic ordinal loss, simply used in the ranking setting.

*5.2.3 LambdaLoss, RankNet and LambdaRank.* We used NDCGLoss2++ of [31], formulated as follows:

$$l(\boldsymbol{s}, \boldsymbol{y}) = -\sum_{y_i > y_j} \log_2 \sum_{\pi} \left( \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \right)^{(\rho_{ij} + \mu\delta_{ij})|G_i - G_j|} H(\pi|\boldsymbol{s})$$

where

$$G_i = \frac{2^{y_i} - 1}{\text{maxDCG}},$$

**Table 1: NDCG@5 test results on WEB30K**

| Loss | Self-attention | MLP |
|---|---|---|
| Ordinal loss | **52.37±0.33** | 48.84±0.42 |
| NDCGLoss 2++ | 51.71±0.40 | **49.15±0.44** |
| LambdaRank | 51.37±0.41 | 48.77±0.38 |
| ListNet | 51.32±0.29 | 47.81±0.36 |
| RMSE | 50.35±0.52 | 48.24±0.52 |
| RankNet | 50.01±0.44 | 47.54±0.47 |
| ListMLE | 49.34±0.59 | 46.98±0.45 |
| | **XGBoost** | |
| `rank:pairwise` | 46.8 | |

**Table 2: Relative percentage NDCG@60 improvement on e-commerce search logs dataset**

| Loss | Self-attention | MLP |
|---|---|---|
| NDCGLoss 2++ | **3.00** | **1.51** |
| LambdaRank | 2.97 | 1.39 |
| ListNet | 2.93 | 1.24 |
| RankNet | 2.68 | 1.19 |
| | **XGBoost** | |
| `rank:pairwise` | 1.83 | |

$$\rho_{ij} = \left| \frac{1}{D_i} - \frac{1}{D_j} \right|,$$
$$\delta_{ij} = \left| \frac{1}{D_{|i-j|}} - \frac{1}{D_{|i-j|+1}} \right|,$$
$$D_i = log_2(1+i)$$

and $H(\pi|\boldsymbol{s})$ is a hard assignment distribution of permutations, i.e.

$$H(\hat{\pi}|\boldsymbol{s}) = 1 \text{ and } H(\pi|\boldsymbol{s}) = 0 \text{ for all } \pi \neq \hat{\pi}$$

where $\hat{\pi}$ is the permutation in which all items are sorted by decreasing scores $\boldsymbol{s}$. Fixed parameter $\mu$ is set to 10.0.

By removing the exponent in $l(\boldsymbol{s}, \boldsymbol{y})$ formula we obtain the RankNet loss function, weighing each score pair identically. Similarly, we may obtain differently weighted RankNet variants by changing the formula in the exponent.

To obtain a LambdaRank formula, replace the exponent with

$$\Delta\text{NDCG}(i,j) = |G_i - G_j|\rho_{ij}.$$

*5.2.4 ListNet and ListMLE.* ListNet loss [9] is given by the following formula:

$$l(\boldsymbol{s}, \boldsymbol{y}) = -\sum_j \text{softmax}(\boldsymbol{y})_j \times \log(\text{softmax}(\boldsymbol{s})_j)$$

In binary version, softmax of ground truth $\boldsymbol{y}$ is omitted for single-click lists and replaced with normalisation by the number of clicks for multiple-click lists.

ListMLE [32] is given by:

$$l(\boldsymbol{s}, \boldsymbol{y}) = -\log P(\boldsymbol{y}|\boldsymbol{s})$$

where

$$P(\boldsymbol{y}|\boldsymbol{s}) = \prod_i^n \frac{\exp(f(x_{y(i)}))}{\sum_{k=i}^n \exp(f(x_{y(k)}))}$$

and $y(i)$ is the index of object which is ranked at position $i$.

## 5.3 Experimental setup

We train both our context-aware ranking models and MLP models on both datasets, using all loss functions discussed in Section 5.2 [2]. We also train XGBoost models with `rank:pairwise` loss similar to the production model of the e-commerce search engine for both datasets. Hyperparameters of all models (number of encoder blocks, number of attention heads, dropout, etc.) are tuned on the validation set of Fold 1 for each loss separately. MLP models are constructed to have a similar number of parameters to context-aware ranking models. For optimisation of neural network models, we use Adam optimiser [19] with the learning rate tuned separately for each model. Details of hyperparameters used can be found in Appendix A. In Section 6 we provide an ablation study of the effect of various hyperparameters on the model's performance.

## 5.4 Results

On WEB30K, models' performance is evaluated using NDCG@5[3], which is the usual metric reported for this dataset. Results are reported in Table 1. On e-commerce search logs, we report a relative percentage increase in NDCG@60 [4] over production XGBoost model, presented in Table 2. We observe consistent and significant performance improvement of the proposed self-attention based model over MLP baseline across all types of loss functions considered. In particular, for ListNet we observe a 7.3% performance improvement over MLP baseline on WEB30K. Note also that the best performing MLP model is outperformed even by the worst-performing self-attention based model on both datasets. We thus observe that incorporating context-awareness into the model architecture has a more pronounced effect on the performance of the model than varying the underlying loss function. Surprisingly, ordinal loss outperforms more established and better-studied losses like ListNet, ListMLE or NDCGLoss 2++ on multi-level relevancy data. In particular, we improve on the previous state-of-the-art result of 51.21 obtained using NDCGLoss 2++ trained using LigthGBM by 2.27%, obtaining 52.37. Another surprising finding is a good performance of models trained with RMSE loss, especially as compared to models trained to optimise RankNet and ListMLE. For comparison with the other methods, we provide results on WEB30K reported in other works in Table 3. For models with multiple variants, we cite the best result reported in the original work. In all tables, boldface is the best value column-wise.

## 5.5 Re-ranking

All experiments on WEB30K described above were conducted in the ranking setting - input lists of items were treated as unordered, thus positional encoding was not used. To verify the effect of positional encoding on the model's performance, we conduct the following

[2]For the clickthrough logs dataset, we used only the losses which can be applied to binary relevance labels.

[3]Expressed as a percentage.

[4]We chose 60 as users' are presented with search results lists of length 60.

[5]Please note that the GSF result was calculated after dropping the approx. 3% queries without any relevant documents, as reported in [2].

**Table 3: NDCG@5 on WEB30K**

| Method | Result |
|---|---|
| GSF [5] | 44.46 |
| DLCM | 45.00 |
| NDCGLoss 2++ (LightGBM) | 51.21 |
| Context-Aware Ranker (this work) | **52.37** |

**Table 4: NDCG@5 on re-ranking task**

| Loss | With PE | Self-attention w/o PE |
|---|---|---|
| Ordinal loss | **52.67** | **52.20** |
| NDCGLoss 2++ | 52.24 | 51.40 |
| RMSE | 51.85 | 50.23 |
| ListNet | 51.77 | 51.34 |
| LambdaRank | 51.51 | 51.22 |
| ListMLE | 50.90 | 49.19 |
| RankNet | 50.58 | 49.90 |

experiments on WEB30K. To avoid information leak, training data[6] is divided into five folds and five XGBoost models are trained, each on four folds. Each model predicts scores for the remaining fold, and the entire dataset is sorted according to these scores.

Finally, we train the same models[7] as earlier on the sorted dataset but use fixed positional encoding. Results are presented in Table 4. As expected, the models are able to learn positional information and demonstrate improved performance over the plain ranking setting.

**Table 5: Ablation study**

| Parameter | Value | Params | WEB30K NDCG@5 |
|---|---|---|---|
| baseline | | 950K | 49.01 |
| $H$ | 1 | 950K | 51.99 |
| | 4 | 950K | 51.96 |
| $N$ | 1 | 250K | 50.33 |
| | 2 | 490K | 51.72 |
| $d_h$ | 64 | 430K | 50.92 |
| | 128 | 509K | 51.31 |
| | 256 | 650K | 51.81 |
| | 1024 | 1540K | **52.11** |
| $p_{\text{drop}}$ | 0.0 | 950K | 42.18 |
| | 0.1 | 950K | 50.36 |
| | 0.2 | 950K | 51.84 |
| | 0.3 | 950K | 51.99 |
| | 0.5 | 950K | 51.32 |
| $l$ | 30 | 950K | 50.48 |
| | 60 | 950K | 51.14 |
| | 120 | 950K | 51.72 |
| | 360 | 950K | 51.89 |

[6]In experiments with positional encoding, we used only Fold 1 of the dataset.

[7]The only loss for which we had to modify the model's hyperparameters as compared to training on the ranking task was NDCGLoss 2++. We observed severe overfitting when training on re-ranking task and thus reduced the number of encoder blocks $N$ from 4 to 2 and hidden dimension $d_h$ form 512 to 256.

## 6 ABLATION STUDY

To gauge the effect of various hyperparameters of self-attention based ranker on its performance, we performed the following ablation study. We trained the context-aware ranker with the ordinal loss on Fold 1 of WEB30K dataset and experimented with a different number $N$ of encoder blocks, $H$ attention heads, length $l$ of longest list used in training, dropout rate $p_{drop}$ and size $d_h$ of hidden dimension. Results are summarised in Table 5. Baseline model (i.e. the best performing context-aware ranker trained with ordinal loss) had the following values of hyperparameters: $N = 4$, $H = 2$, $l = 240$, $p_{drop} = 0.4$ and $d_h = 512$. We observe that a high value of dropout is essential to prevent overfitting. Even though it is better to use multiple attention heads as opposed to a single attention head, using too many results in performance degradation. A similar statement is true of the hidden dimension - generally, larger hidden dimension yields better performance, but setting $d_h$ to 1024 does not outperform the model trained with $d_h = 512$, which established the new state-of-the-art. Finally, stacking multiple encoder blocks increases performance. However, we did not test the effect of stacking more than 4 encoder blocks due to GPU memory constraints.

## 7 CONCLUSIONS

In this work, we addressed the problem of constructing a context-aware scoring function for learning to rank. We adapted the self-attention based Transformer architecture from the neural machine translation literature to propose a new type of scoring function for LTR. We demonstrated considerable performance gains of proposed neural architecture over MLP baselines across different losses and types of data, both in ranking and re-ranking setting. In particular, we established the new state-of-the-art performance on WEB30K. These experiments provide strong evidence that the gains are due to the ability of the model to score items simultaneously. As a result of our empirical study, we observed the strong performance of models trained to optimise ordinal loss function. Such models outperformed models trained with well-studied losses like LambdaLoss or LambdaMART, which were previously shown to provide tight bounds on IR metrics like NDCG. On the other hand, we observed the surprisingly poor performance of models trained to optimise RankNet and ListMLE losses. In future work, we plan to investigate the reasons for both good and poor performance of the aforementioned losses, in particular, the relation between ordinal loss and NDCG.

## REFERENCES


[1] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. 2018. Learning a Deep Listwise Context Model for Ranking Refinement. *CoRR* abs/1804.05936 (2018). arXiv:1804.05936 http://arxiv.org/abs/1804.05936

[2] Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Mike Bendersky, and Marc Najork. 2019. Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks. In *Proceedings of the 5th ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR)*.

[3] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *ArXiv* abs/1607.06450 (2016).

[4] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[5] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Huai-hsin Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. 2018. Seq2Slate: Re-ranking and Slate Optimization with RNNs. *CoRR* abs/1810.02019 (2018). arXiv:1810.02019 http://arxiv.org/abs/1810.02019

[6] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to Rank Using Gradient Descent. In *Proceedings of the 22Nd International Conference on Machine Learning (ICML '05)*. ACM, New York, NY, USA, 89–96. https://doi.org/10.1145/1102351.1102363

[7] Christopher J. Burges, Robert Ragno, and Quoc V. Le. 2007. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman (Eds.). MIT Press, 193–200. http://papers.nips.cc/paper/2971-learning-to-rank-with-nonsmooth-cost-functions.pdf

[8] Christopher J. C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report. Microsoft Research. http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf

[9] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. ACM, New York, NY, USA, 129–136. https://doi.org/10.1145/1273496.1273513

[10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[11] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long Short-Term Memory-Networks for Machine Reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, 551–561. https://doi.org/10.18653/v1/D16-1053

[12] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. ACM, New York, NY, USA, 65–74. https://doi.org/10.1145/3077136.3080832

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 770–778.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[15] Saratchandra Indrakanti, Svetlana Strunjas, Shubhangi Tandon, and Manojkumar Rangasamy Kannadasan. 2019. Exploring the Effect of an Item's Neighborhood on its Sellability in eCommerce. arXiv:cs.IR/1908.03825

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446. https://doi.org/10.1145/582415.582418

[17] Ray Jiang, Sven Gowal, Yuqiu Qian, Timothy Mann, and Danilo J. Rezende. 2019. Beyond Greedy Ranking: Slate Optimization via List-CVAE. In *International Conference on Learning Representations*. https://openreview.net/forum?id=r1xX42R5Fm

[18] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 3146–3154. http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[19] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[20] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (March 2009), 225–331. https://doi.org/10.1561/1500000016

[21] Andrew Y. Ng. 2004. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML '04)*. ACM, New York, NY, USA, 78–. https://doi.org/10.1145/1015330.1015435

[22] Zhenxing Niu, Mo Zhou, Le Wang, and Xinbo Gao. 2016. Ordinal Regression with Multiple Output CNN for Age Estimation. 4920–4928. https://doi.org/10.1109/CVPR.2016.532

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[24] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, and Wenwu Ou. 2019. Personalized Re-ranking for Recommendation. arXiv:cs.IR/1904.06813

[25] Tao Qin and T. M. Liu. 2013. Introducing LETOR 4.0 Datasets. *ArXiv* abs/1306.2597 (2013).

[26] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3483–3491. http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958. http://dl.acm.org/citation.cfm?id=2627435.2670313

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[29] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2692–2700. http://papers.nips.cc/paper/5866-pointer-networks.pdf

[30] Ellen M. Voorhees. 1999. The TREC-8 Question Answering Track Report. In *In Proceedings of TREC-8*. 77–82.

[31] Xuanhui Wang, Cheng Li, Nadav Golbandi, Mike Bendersky, and Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *Proceedings of The 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*. 1313–1322.

[32] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. ACM, New York, NY, USA, 1192–1199. https://doi.org/10.1145/1390156.1390306

# A EXPERIMENTS DETAILS

In Tables 6 and 7 we provide hyperparameters used for all models reported in Table 1. Models trained on WEB30K were trained for 100 epochs with the learning rate decayed by 0.1 halfway through the training. On e-commerce search logs, we trained the models for 10 epochs and decayed the learning rate by 0.1 after 5-th epoch. The meaning of the columns in Table 6 is as follows: $d_{f_c}$ is the dimension of the linear projection done on the input data before passing it to the context-aware ranker, $N$ is the number of encoder blocks, $H$ is the number of attention heads, $d_h$ is the hidden dimension used throughout computations in encoder blocks, $lr$ is the learning rate, $p_{\text{drop}}$ is the dropout probability and $l$ is the list length (lists of items were either padded or subsampled to that length). The last column shows the number of learnable parameters of the model.

In Table 7, *Hidden dimensions* column gives dimensions of subsequent layers of MLP models. The remaining columns have the same meaning as in the other table.

**Table 6: Details of hyperparameters used in self-attentive models**

| **Loss** | $d_{fc}$ | $N$ | $H$ | $d_{\text{h}}$ | lr | $p_{\text{drop}}$ | $l$ | \|params\| |
|---|---|---|---|---|---|---|---|---|
| **WEB30K** | | | | | | | | |
| NDCGLoss 2++ | 128 | 4 | 4 | 512 | 1e-3 | 0.3 | 240 | 811K |
| LambdaRank | 128 | 4 | 4 | 512 | 1e-3 | 0.3 | 240 | 811K |
| ListMLE | 256 | 4 | 4 | 512 | 1e-3 | 0.3 | 240 | 2.14M |
| ListNet | 128 | 4 | 4 | 512 | 1e-3 | 0.3 | 240 | 811K |
| Ordinal loss | 144 | 4 | 2 | 512 | 1e-3 | 0.4 | 240 | 949K |
| RankNet | 144 | 4 | 2 | 512 | 1e-3 | 0.3 | 240 | 949K |
| **E-commerce** | | | | | | | | |
| NDCGLoss 2++ | 128 | 2 | 2 | 128 | 1e-3 | 0.0 | 60 | 206K |
| LambdaRank | 128 | 2 | 2 | 128 | 1e-3 | 0.0 | 60 | 206K |
| ListNet | 128 | 2 | 2 | 128 | 2e-3 | 0.0 | 60 | 206K |
| RankNet | 128 | 2 | 2 | 128 | 2e-3 | 0.0 | 60 | 206K |

**Table 7: Details of hyperparameters used in MLP models**

| **Loss** | Hidden dimensions | lr | $p_{\text{drop}}$ | $l$ | \|params\| |
|---|---|---|---|---|---|
| **WEB30K** | | | | | |
| all losses | [256, 512, 1024, 512, 256] | 1e-3 | 0.3 | 240 | 1.35M |
| **E-commerce** | | | | | |
| NDCGLoss 2++ | [256, 384, 256] | 1e-3 | 0.0 | 60 | 210K |
| LambdaRank | [256, 384, 256] | 1e-3 | 0.0 | 60 | 210K |
| ListNet | [256, 384, 256] | 1e-4 | 0.0 | 60 | 210K |
| RankNet | [256, 384, 256] | 1e-4 | 0.0 | 60 | 210K |