NeuralNDCG: Direct Optimisation of a Ranking Metric via Differentiable Relaxation of Sorting

Przemysław Pobrotyn Radosław Białobrzeski

Machine Learning Research at Allegro.pl

allegro ML Research

Learning to Rank

Problem formulation

Learning to Rank (LTR) is concerned with optimising the global ordering of a list of items according to their utility to the user. It is ubiquitous in industrial applications - whenever one needs to optimise the order of a list of search results, LTR algorithms provide a solution. To frame it as a machine learning problem, one needs a dataset of search results together with their utility to the users. The utility can be expressed as either graded relevance (human-annotated, on a scale of, say, 1 to 5) or binary relevance (usually, clickthrough logs of a live search engine). Given such a dataset, a ranking function can be learned to produce a desired permutation of items. Most LTR algorithms fall into **score & sort** type: using the training data, a scoring function f is learned which scores items, either individually or by looking at the entire list at once. Items are then sorted in descending order of the scores. To learn f, typically one uses a pointwise, pairwise or a listwise ranking loss function. Popular examples include RankNet [1] or ListNet [2].

Evaluation metric

To evaluate the ranking function, we need a metric that emphasises relevant items being placed on top of the list.

The commonly used metric in LTR is (Normalised) Discounted Cumulative Gain.

$$NDCG_k = N_k^{-1} \sum_{j=1}^{k} g(r_j) d(j)$$

where r_j denotes the relevance of the item ranked at j-th position, $g(r_j)$ denotes a gain function, $d(j)$ denotes a discount function, and N_k denotes the maximum value of $\sum_{j=1}^{k} g(r_j)d(j)$, which is attained when documents are ranked in the descending order of ground truth relevance labels. Typically, the gain function is $g(r_j) = 2^{r_j} - 1$ and the discount function is $d(j) = 1/\log_2(j+1)$.

Limitations

Since NDCG relies on the sorting operator, its derivative is either zero or undefined. Thus, it is unsuitable for gradient based optimisation. Popular ranking loss functions are only a proxy to the evaluation metric, creating a mismatch between training and evaluation.

Our solution

We propose a novel differentiable approximation to NDCG, called NeuralNDCG. We introduce two variants of the proposed loss function and demonstrate empirically its effectiveness.

Contributions

- We introduce **NeuralNDCG**, a novel smooth approximation of NDCG based on a differentiable relaxation of the sorting operator.
- We evaluate our approach on **Web30K** and **Istella**, benchmark datasets for learning to rank and show favourable performance as compared to baselines
- We plan to open-source our implementation of NeuralNDCG as part of **allRank, a PyTorch framework for reproducible, neural LTR**

Loss formulation

Sorting relaxation

Our novel NDCG approximation relies on a differentiable approximation of the sorting operator. To that end, we use NeuralSort [3].

Recall that sorting a list of scores $\boldsymbol{s}$ is equivalent to left-multiplying a column vector of scores by the permutation matrix $P_{\texttt{sort}(\boldsymbol{s})}$ induced by permutation $\texttt{sort}(\boldsymbol{s})$ sorting the scores. Thus, in order to approximate the sorting operator, it is enough to approximate the induced permutation matrix. In [3], the permutation matrix is approximated via a unimodal row stochastic matrix $\widehat{P}_{\texttt{sort}(\boldsymbol{s})}(\tau)$ given by:

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

where $A_{\boldsymbol{s}}$ is the matrix of absolute pairwise differences of elements of $\boldsymbol{s}$ such that $A_{\boldsymbol{s}}[i,j] = |s_i - s_j|$, $\mathbb{1}$ denotes the column vector of all ones and $\tau > 0$ is a temperature parameter controlling the accuracy of approximation. For brevity, for the remainder of the work we refer to $\widehat{P}_{\texttt{sort}(\boldsymbol{s})}(\tau)$ simply as $\widehat{P}$.

NeuralNDCG in two flavours

Using the approximation of the sorting operator given by NeuralSort, we define two approximations to NDCG. In the first one, dubbed **NeuralNDCG**, we first approximately sort the relevancies using $\widehat{P}$ and mutliply the result by the logairthmic position discounts. The summation is done over the ranks of documents. In the other variant, called **NeuralNDCG Transposed** ($\mathrm{NeuralNDCG}^T$), the approximate permutation matrix $\widehat{P}$ is transposed so that we use it to approximately sort the discounts in the order of the documents, mutliply the result by documents relevancies and then sum over the documents, not ranks. Essentially both variants differ in the order they carry out matrix multiplications, needing to transpose $\widehat{P}$ in $\mathrm{NeuralNDCG}^T$ as a result. The formulae for both loss functions are as follows:

$$\mathrm{NeuralNDCG}_k(\tau)(\boldsymbol{s}, \boldsymbol{y}) = N_k^{-1} \sum_{j=1}^{k} (\mathrm{scale}(\widehat{P}) \cdot g(\boldsymbol{y}))_j \cdot d(j) \tag{2}$$

$$\mathrm{NeuralNDCG}^T{}_k(\tau)(\boldsymbol{s}, \boldsymbol{y}) = N_k^{-1} \sum_{i=1}^{n} g(\boldsymbol{y}_i) \cdot (\mathrm{scale}(\widehat{P}^T) \cdot \boldsymbol{d})_i \tag{3}$$

where N_k^{-1} is the maxDCG at k-th rank, $\mathrm{scale}(\cdot)$ is Sinkhorn scaling and $g(\cdot)$ and $d(\cdot)$ are their gain and discount functions. In the second formula, $\boldsymbol{d}$ is the vector of logarithmic discounts per rank set to 0 for ranks $j > k$.

Properties

In computation of $\widehat{P}^T$, 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. In fact, it is not difficult to demonstrate that:

$$\lim_{\tau \to 0} \widehat{P}_{\texttt{sort}(\boldsymbol{s})}(\tau) = P_{\texttt{sort}(\boldsymbol{s})} \tag{4}$$

Thus, as the temperature approaches zero, NeuralNDCG approaches true NDCG in both its variants. See Figure 1 for examples of the effect of the temperature on the accuracy of the approximation.

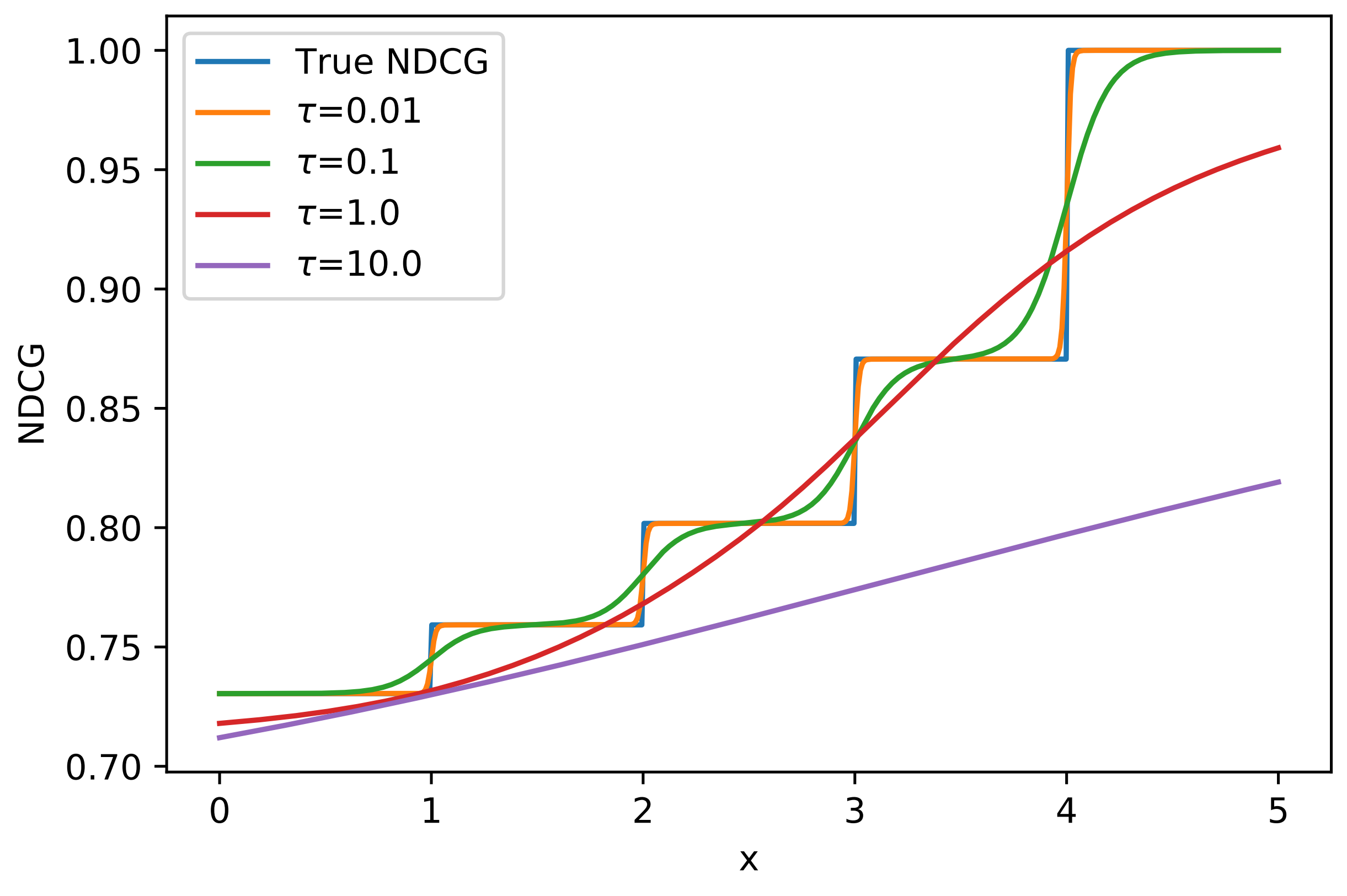

Figure 1. Given ground truth $\boldsymbol{y} = [1,2,3,4,5]$ and a list of scores $\boldsymbol{s} = [1,2,3,4,x]$, we vary the value of the score x and plot resulting NDCG induced by the scores along with NeuralNDCG for different temperatures τ.

Experimental results

We train a Transformer based, Context-Aware Ranker [4] with the proposed loss functions and verify their performance on two benchmakr LTR datasets, WEB30K and Istella. Results are presented in the Table 1 below.

Table 1. Test NDCG on Web30K and Istella. Boldface is the best performing loss column-wise and † next to a result means there is no statistically significant difference between that result and the best result in the column, according to t-test at level of 0.05.

Loss	WEB30K		Istella	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
NeuralNDCG@5	49.93	51.87	65.46†	70.09
NeuralNDCG@10	50.31†	52.33†	65.97†	70.90†
NeuralNDCG@max	50.42†	52.36†	65.55†	70.50
NeuralNDCG^T@5	50.29†	52.11	65.32	69.90
NeuralNDCG^T@10	50.27†	52.26†	65.81†	70.83†
NeuralNDCG^T@max	50.66†	52.74†	65.67†	70.51
ApproxNDCG	49.35	51.14	63.28	68.08
ListNet	50.51†	52.67†	65.76†	70.86†
ListMLE	49.19	51.36	60.25	66.42
RankNet@5	48.53	50.18	64.70	69.00
RankNet@10	50.02	51.88	65.83†	70.90†
RankNet@max	49.30	51.62	64.59	70.40
LambdaRank@5	48.20	49.77	63.74	67.82
LambdaRank@10	48.98	50.93	65.05	69.54
LambdaRank@max	**50.96**	**53.00**	65.90†	71.09†
RMSE	50.07†	51.97	**66.01**	**71.15**
LambdaMART	46.80	49.17	61.04	65.74

allRank - for all your ranking needs!

This reasearch was carried out in allRank, an open source PyTorch framework featuring:

- out-of-the-box support for feed-forward and self-attention based architectures
- implementation of most popular pointwise, pairwise and listwise losses
- easy extensibility with your fancy new ranking loss function
- support for WEB30K and Istella, the standard ranking datasets

We plan to open source our implementation of NeuralNDCG in the near future.
Visit github.com/allegro/allRank for code and a getting-started guide.
Contributions are welcome!

References

[1] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender.
Learning to rank using gradient descent.
In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[2] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li.
Learning to rank: From pairwise approach to listwise approach.
In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, New York, NY, USA, 2007. ACM.

[3] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon.
Stochastic optimization of sorting networks via continuous relaxations.
In *International Conference on Learning Representations*, 2019.

[4] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar.
Context-aware learning to rank with self-attention.
In *SIGIR eCom '20*, Virtual Event, China., 2020.