Report Homework 3 Information Retrieval

Tom Lotze
11161159
University of Amsterdam
Tom.Lotze@gmail.com

Berend Jansen
11051949
University of Amsterdam
bereeend@gmail.com

Ruth Wijma
10815252
University of Amsterdam
rlwijma@gmail.com

1 OFFLINE LEARNING TO RANK

2 POINTWISE LTR

All the models can be found in the pointwise_ltr directory. The description of functions per file can be found in the README file.

Optimal parameters. Various searches were run to find the optimal parameters for the model. The script used for the grid-search is pointwise_tuning.sh. The model is evaluated on nDCG on the test set after convergence in training. This convergence is based on the average nDCG on validation set per epoch (tested every 100 batches; 924 batches per epoch). If the average nDCG stops decreasing the training is stopped. The following parameters are tested and compared (40 parameter settings in total):

- Hidden units (in listwise notation): [512, 512, 512, 512], [128, 128, 128], [512, 128, 8], [256, 10], [256]
- Optimizer: Adam and SGD.
- Learning rate: 0.001, 0.005, 0.01, 0.05 (all learning rates were tested for both optimizers).

The best performing parameters are as follows: Hidden units; [256, 10], learning rate 0.001 and optimizer Adam. From now on this will be referred to as the "best model".

Performance on test set. The best model as defined above was evaluated on the test set, and the obtained scores are shown in table 1.

Metric	Score	STD
dcg	19.021	14.429
dcg@03	8.495	7.422
dcg@05	10.431	8.329
dcg@10	13.437	9.724
dcg@20	16.232	11.518
ndcg	0.824	0.148
ndcg@03	0.636	0.290
ndcg@05	0.662	0.256
ndcg@10	0.719	0.222
ndcg@20	0.770	0.187
precision@01	0.558	0.497
precision@03	0.384	0.307
precision@05	0.309	0.240
precision@10	0.226	0.171
precision@20	0.151	0.121
recall@01	0.238	0.321
recall@03	0.415	0.373
recall@05	0.523	0.376
recall@10	0.703	0.345
recall@20	0.859	0.253
relevant rank	13.252	14.345
relevant rank per query	51.562	95.332

Table 1: Results on test set of the best pointwise model as defined in question 2

Analysis Question 2.1

The training loss and nDCG on the validation during training are shown in figure 1. The graph shows quite fast convergence (after 7 epochs). Moreover, the nDCG increases fast after a few batches, to afterwards slowly increase until convergence, at which point the nDCG is around 0.82. The training loss steadily decreases during training, and does not increase further if training is continued (data not shown). As expected, the loss and nDCG are inversely correlated throughout training.

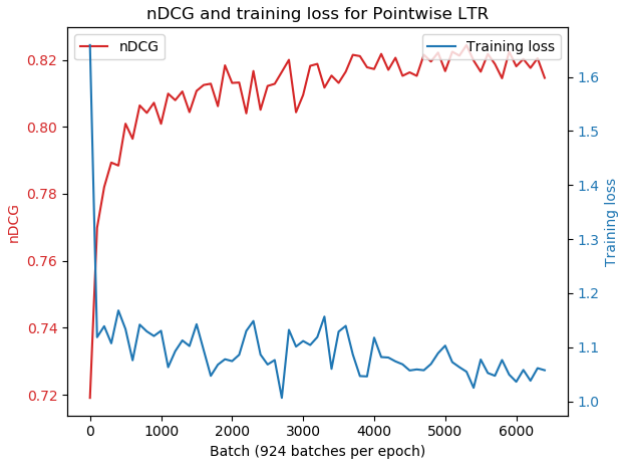

Figure 1: Training loss and nDCG on validation set for the best performing model described in 2

Analysis Question 2.2

The distributions of relevance classes are shown in figure 2 (validation set), 3 (test set) and 4 (predicted labels on test set). The distributions are very similar, with high frequencies of low relevance classes 1, 2, and 3, and low frequencies of the very relevant classes 4 and 5.

The (small) difference between the actual and predicted labels of the test set is the fraction of relevance class 3. We can say that our model underestimates the number of documents in this class. This could be explained by the relative low frequency of these high

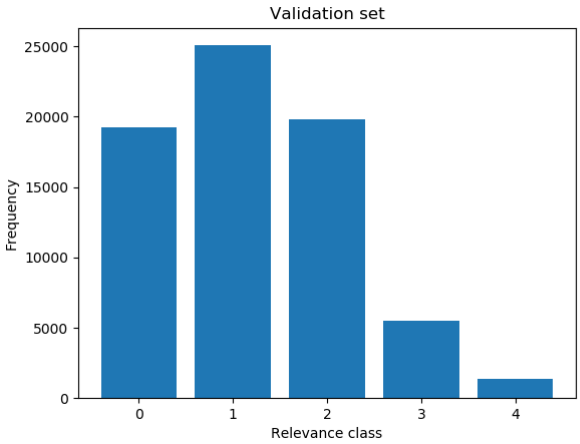

Figure 2: Distribution of relevance scores in validation set

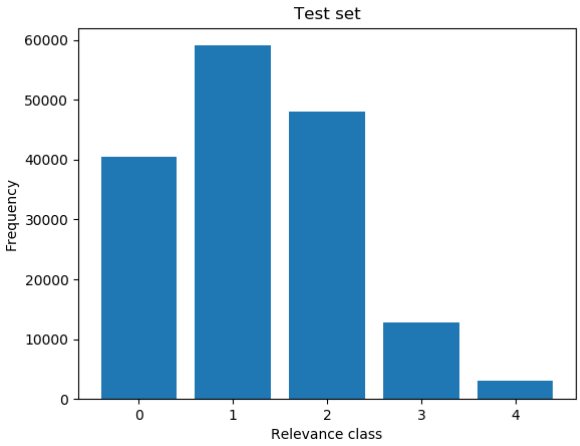

Figure 3: Distribution of relevance scores in test set

relevance classes, and hence the low expected "payoff" (decrease in loss) generated by predicting these classes.

3 PAIRWISE LTR

3.1 Normal RankNet

The normal RankNet module class with corresponding training scripts can be found in the pairwise_ltr folder. The early stopping mechanism is the same as for the pointwise model.

3.1.1 Optimal parameters. We performed one search on the hyperparameters of the RankNet model. We tested for the following settings:

- Hidden layer size: 128, 256, 512
- Learning rate: 0.001, 0.01, 40.005, 0.05

The model that performed best had a hidden layer size of 128 and a learning rate of 0.005.

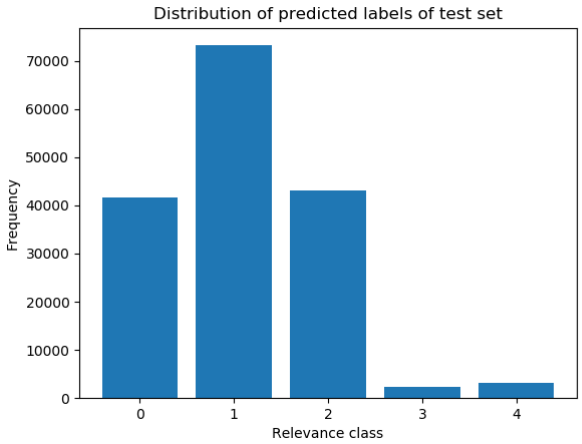

Figure 4: Distribution of relevance scores in predicted labels on test set. The model used is the best model as described in 2

Metric	Score	STD
dcg	19.154	14.214
dcg@03	8.596	7.001
dcg@05	10.6	7.895
dcg@10	13.642	9.378
dcg@20	16.455	11.248
ndcg	0.837	0.14
ndcg@03	0.659	0.281
ndcg@05	0.685	0.247
ndcg@10	0.738	0.212
ndcg@20	0.788	0.177
precision@01	0.579	0.494
precision@03	0.405	0.3
precision@05	0.325	0.237
precision@10	0.232	0.169
precision@20	0.154	0.121
recall@01	0.248	0.325
recall@03	0.447	0.374
recall@05	0.559	0.374
recall@10	0.732	0.332
recall@20	0.88	0.229
relevant rank	12.516	13.694
relevant rank per query	48.7	93.402

Table 2: RankNet results on test set.

3.1.2 Performance on test set. The performance of the best model as described above on the test set is shown in table 2.

3.2 Sped up RankNet

3.2.1 Optimal parameters. The same parameter settings were tested as in the non-sped-up version of RankNet, see above. The model that performed best has a hidden layer size of 128 and

dcg	18.852	13.876
dcg@03	8.192	6.540
dcg@05	10.256	7.478
dcg@10	13.287	8.933
dcg@20	16.122	10.821
ndcg	0.828	0.140
ndcg@03	0.642	0.279
ndcg@05	0.672	0.244
ndcg@10	0.727	0.211
ndcg@20	0.778	0.176
precision@01	0.522	0.500
precision@03	0.385	0.292
precision@05	0.319	0.230
precision@10	0.229	0.166
precision@20	0.152	0.118
recall@01	0.220	0.314
recall@03	0.430	0.373
recall@05	0.553	0.376
recall@10	0.722	0.336
recall@20	0.876	0.233
relevant rank	12.841	13.827
relevant rank per query	49.966	95.264

Table 3: Results on test set of the Sped-up RankNet model with tuned parameters

a learning rate of 0.005, which corresponds to the best settings for the default RankNet.

3.2.2 Performance on test set. The scores of the best model as described above are shown in table 3.

Analysis Question 3.1

In figure 5 the nDCG on the validation set for both the default RankNet variant, as well as the sped-up RankNet, both using the best parameters found in the parameter search (see 3.2.1). The figures show a comparable convergence rate. For both models the training stopped early, for the default RankNet after 5 epochs, for the Sped-up RankNet after 8 epochs. However, comparing the figures (5), the plots do not support the claim that one model converges faster than the other. As we can see, both reach high nDCG within 10000 batches and do not improve much after that. The default RankNet shows almost no improvements after the first epoch anymore, whereas the Sped-up RankNet does increase slowly throughout training. The difference in actual timing between the original and the sped-up variant is marginal.

Analysis Question 3.2

The nCDG and ARR of the best Sped-up RankNet model are plotted in figure 6. We can see that there is no significant difference between the two metrics in terms of which one is better optimized, hence we cannot draw a conclusion about that.

4 LISTWISE LTR WITH LAMBDARANK

The LambdaRank model can be found in the LambdaRank_ltr folder. The model uses a shallow network to compute the scores

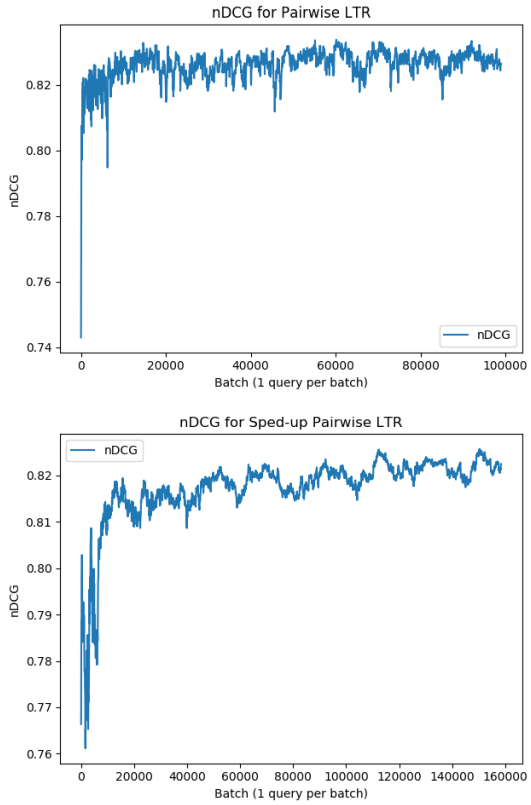

Figure 5: Convergence rate of RankNet (top) and sped-up RankNet (bottom)

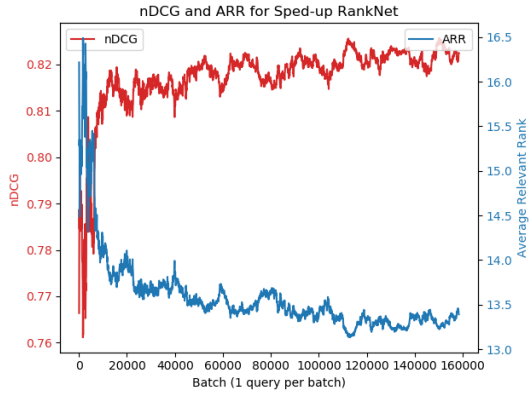

Figure 6: nDCG and ARR for the best Sped-up RankNet model

for each document. These scores are used to compute the λ-values. These values are multiplied by the absolute change in the IR metric as described in the assignment.

We trained two models, one model optimizing the nDCG and one model optimizing the ERR. For both models we ran a hyperparameter search over the following settings:

Model	nDCG	ERR
nDCG-LambdaRank	0.8294	0.4394
ERR-LambdaRank	0.8184	0.4344

Table 4: Performance of both variants of the LambdaRank

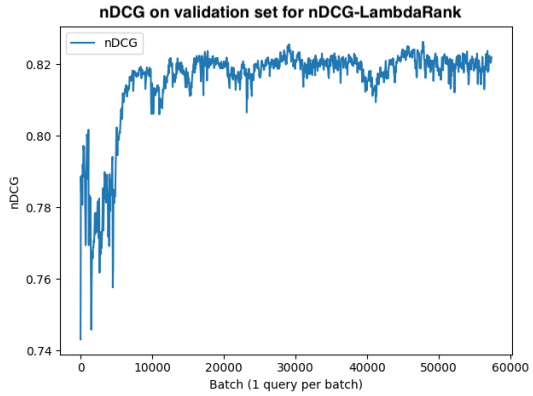

Figure 7: nDCG on validation set for the best LambdaRank model

- Hidden layer size: 128, 256, 512
- Learning rate: 0.001, 0.01, 40.005, 0.05

The optimal settings for the models were 128 hidden units and a learning rate of 0.005. The performance on the test set for both variants is shown in table 4.

The models were trained until convergence, with a maximum of 10 epochs. During training, the models were evaluated on the test set every 1000 batches, where 1 batch consists of 1 query.

Analysis Question 4.1

We retrained the best LambdaRank performing model (nDCG, 128 hidden units, learning rate 0.005) and evaluated on the validation set every 50 batches. The nDCG on the validation set is plotted in figure 7. The plot shows a steep increase in nDCG during the first epoch, yet slower than the previous methods. The drop in nDCG after about 2000 batches is remarkable, but the model recovers fast after that, and does not really improve after 10000 batches. The drop could be explained by the fact that 1 batch consists of all the docs for 1 query. It could be that the previous 1000 batches were slightly different than the average query in the validation set, pushing the model to worse performance on the validation set.

Analysis Question 4.2

Table 5 shows the results of the best performing models on the test set. We report the nDCG and ERR. All the models score decently well and the results are somewhat as expected in terms of nDCG. However, we expected the listwise methods to perform the best, but the default pairwise method actually performs best. Nevertheless, the differences in performance are small, smaller then we expected, given that the Pointwise and pairwise methods do not directly optimize the ranking.

Metric	nDCG	ERR
Pointwise	0.824 (0.148)	-
Pairwise (regular)	0.8367 (0.14)	0.427 (0.2694)
Pairwise (Sped-up)	0.828 (0.140)	0.358 (0.2149)
Listwise (nDCG)	0.829 (0.142)	0.439 (0.2884)
Listwise (ERR)	0.818 (0.146)	0.434 (0.295)

Table 5: Results we obtained when we evaluated the models on the test set. The mean result is given, along with the standard deviation in parentheses

The ERR shows similar trends. All models score quite similar, except for the sped-up RankNet, which has a significant lower ERR score. However, all of the standard deviations are very high, so drawing conclusions from these data is hard. The reason for the outlier for the Sped-up RankNet is unknown.

THEORY QUESTIONS

Theory Question 1.1

The feature vector should not be too small, since it needs to encode information about both the query and the document. It should also be not too big, since this makes it harder to model any signal. An approach would be to use a model that encodes the information in the document, such as Doc2Vec [1], which could also be used to encode the query. The Learning to rank model can then estimate the relevance from these concatenated document-query embeddings.

Theory Question 1.2

In Offline LTR, you use annotated data to train models, and these annotated datasets are time-consuming and costly to create, meaning that these are often small in size. This makes it harder to train models, since the models need to be more efficient with the training data.

Theory Question 1.3

Learning to rank differs in objective from ordinal regression/ classification since it sees the problem as trying to create an optimal ranking, as opposed to predicting the individual relevance labels. The predicted relevance labels themselves are not that important, the relative ordering between them is.

Theory Question 1.4

It is not possible to directly optimize any given metric, since we need the derivatives to update our model parameters. E.g. both ERR and nDCG are non-continuous and non-differentiable with respect to the inputs.

Theory Question 2.1

Pointwise LTR does not directly optimize the ranking, but the relevance judgements of query-document pairs. This means there are examples in which the loss actually decreases (and the model "thinks"

it performs better), but the actual ranking gets worse. A simple example might be where we have 5 documents, one of which is relevant. Let ranking A be a ranking in which the only relevant document is placed at the top with the highest relevance score, but all the irrelevant documents get a relatively high score too. Let ranking B be a ranking in which all documents are scored very low, but the only relevant document is scored the lowest. While ranking A should be scored better (the relevant document is placed at the top), ranking B might have a lower loss. Thus, a lower loss does not necessarily imply a better ranking, which is problematic in optimizing those rankings.

Theory Question 3.1

The complexity of training the normal RankNet is $O(m*n^2)$, where m is the number of training queries n the maximum number of documents per query. This number is squared because we have at max n^2 pairs per query.

Theory Question 3.2

The complexity of the Sped-up RankNet is reduced, as not each pair is back-propagated, but first aggregated into a lambda, and then back-propagated. The complexity is still in the order of n^2, but computing the derivative is cheaper.

Theory Question 3.3

The key issue is that the model assumes every pair is equally important, which is actually not the case. The order of the top 10 documents for example, is much more important than the ordering of the documents after position 10. The consequence of this is that rankings that intuitively are better score lower according to the pairwise algorithm. For example take 2 rankings with 2 relevant documents out of 10 documents, in which the positions of the relevant documents are 1, 10 and 5, 6 respectively. The second ranking with the relevant documents in the middle has more pairs correct and is considered the better ranking by the model. However, intuitively, the ranking with a relevant document at position 1 should be better.

Theory Question 4.1

λ_{ij} represents the change in the cost function between document i and j when we change output i, i.e. $\hat{y}_i$ of the model output. This means that λ_{ij} forces $\hat{y}_i$ down if λ_{ij} is positive and pushes it up when it is negative. In the end, only the sum of forces on a document i matters, by taking the final gradient for score s_i to be $\sum_j \lambda_{ij}$.

Theory Question 4.2

We would argue that LambdaRank, while officially being a Listwise LTR method, still is a Pairwise LTR method, since the loss function still only considers pairs of predictions. The loss is based on all documents retrieved for the query by considering all possible pairs. This means that it is in essence just a Pairwise LTR method. Using the change in nDCG or ERR only scales the gradient vector.

REFERENCES

[1] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.