Pretrained deep models outperform GBDTs in Learning-To-Rank under label scarcity

Charlie Hou [1] **Kiran Thekumparampil** [2] **Michael Shavlovsky** [2] **Giulia Fanti** [1] **Yesh Dattatreya** [2] **Sujay Sanghavi** [3]

Abstract

While deep learning (DL) models are state-of-the-art in text and image domains, they have not yet consistently outperformed Gradient Boosted Decision Trees (GBDTs) on tabular Learning-To-Rank (LTR) problems (Qin et al., 2021). Most of the recent performance gains attained by DL models in text and image tasks have used unsupervised pretraining (Devlin et al., 2018; Chen et al., 2020a), which exploits orders of magnitude more unlabeled data than labeled data. To the best of our knowledge, unsupervised pretraining has not been applied to the LTR problem, which often produces vast amounts of unlabeled data. In this work, we study whether unsupervised pretraining can improve LTR performance over GBDTs and other non-pretrained models. Using simple design choices–including SimCLR-Rank, our *ranking-specific* modification of SimCLR (Chen et al., 2020a) (an unsupervised pretraining method for images)–we produce pretrained deep learning models that soundly outperform GBDTs (and other non-pretrained models) in the case where labeled data is vastly outnumbered by unlabeled data. We also show that pretrained models also often achieve significantly better robustness than non-pretrained models (GBDTs or DL models) in ranking outlier data.

1. Introduction

The learning-to-rank (LTR) problem aims to train a model to rank a set of items according to their relevance or user preference (Liu, 2009). An LTR model is typically trained on a dataset of queries and associated *query groups* (i.e., a set of potentially relevant *documents* or *items* per query), as well as an associated (generally incomplete) ground truth ranking of the items in the query group. The model is trained to output a ranking of documents or items in a query group, given a query. LTR is a core problem in many real world applications—most notably in search contexts including Bing web search (Qin and Liu, 2013), Amazon product search (Yang et al., 2022), and Netflix movie recommendations (Lamkhede and Kofler, 2021).

In many applications of LTR, models take as input *tabular features*—numerical or categorical features—of queries and documents (Chapelle and Chang, 2011; Qin and Liu, 2013; Lucchese et al., 2016). Today, identifying deep models that can achieve parity with gradient boosted decision trees (GBDTs) (Friedman, 2001) over tabular features is considered a success (Jeffares et al., 2023; Qin et al., 2021), in contrast to deep models' success in domains like text (Devlin et al., 2018) and images (He et al., 2016).

Recent breakthroughs in modeling non-tabular data like text and images have been driven by unsupervised pretraining (or self-supervised pretraining) of deep neural networks (Devlin et al., 2018; Chen et al., 2020a), followed by supervised finetuning. Models that are pretrained in this way can perform significantly better than models that were only trained on existing labeled data. The remarkable success of SSL in the image and text domains over plain supervised deep learning appears to arise in part from two factors: (1) there exist large, available sources of unlabeled text and image data, and (2) self-supervised models are able to take advantage of unlabeled data.

It is easy to find settings in LTR where the amount of unlabeled data outnumbers labeled data by orders of magnitude. For example, in search ranking problems, users often search for something but may not click or purchase any link or item, which leads to a query group (i.e. a search and the list of results) that has no supervision signal or label for the relevance of the items.

A natural question is whether deep models can outperform tabular methods like GBDTs on the LTR problem by making use of unsupervised pretraining. Note that GBDTs (to

[1] Carnegie Mellon University [2] Amazon (work does not relate to their position here) [3] University of Texas, Austin. Correspondence to: Charlie Hou <charlieh@andrew.cmu.edu>.

The Many Facets of Preference Learning Workshop at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023. Copyright 2023 by the author(s).

the best of our knowledge) are unable to make use of unsupervised pretraining. In this work, we show that the answer to this question is yes, but so far only in extreme cases of label scarcity (i.e. unlabeled data greatly outnumbers labeled data). We base our empirical conclusions off of experiments on three well-known public datasets for ranking: MSLR-WEB30K (Qin and Liu, 2013), Yahoo (Chapelle and Chang, 2011), and Istella (Lucchese et al., 2016).

Contributions. We make four contributions in this work.

- **How to pretrain for LTR?** We first adapt two unsupervised pretraining methods to the LTR setting: SimCLR (Chen et al., 2020a) and SimSiam (Chen and He, 2021). Both are domain-agnostic methods that are strong baselines in the image domain.
- **Empirical success of pretraining.** We evaluate these unsupervised pretraining approaches on three benchmark datasets in the LTR space: MSLR-30k (Qin and Liu, 2013), Yahoo Set1 (Chapelle and Chang, 2011) and Istella-S (Lucchese et al., 2016), which we modify for the unsupervised pretraining setting by not using 99.9% of the labels. We find that pretrained models, under the right training settings, outperform non-pretrained DL and GBDT models across all datasets.
- **Training choices recommendations.** We find that unlike in the image domain (where SimSiam and SimCLR were proposed), linear probing—where one learns a linear model on top of a frozen pretrained model—cannot be used for finetuning and gives poor results. Further, typical training choices for training neural ranking methods (as found in the PT-ranking framework (Yu, 2020) and in neural ranking papers (Qin et al., 2021; Bruch et al., 2019)) can cause instability when pretraining. In this paper, we identify training settings appropriate for unsupervised pretraining in LTR, which include our *ranking-aware* modification to SimCLR, SimCLR-rank.
- **Robustness measurement for LTR.** LTR models are commonly evaluated on ranking metrics (e.g. normalized discounted cumulative gain a.k.a. NCDG) uniformly averaged across test queries (Burges, 2010). However, uniform average gives no insight into their behavior on *outlier queries*. We propose measuring the average NDCG on the outlier queries of a dataset (we call this robust-NDCG). Using this, we show that pretrained methods have significantly better robustness than non-pretrained ones. We suggest that robustness metrics should be studied when evaluating any LTR model.

2. Background on Learning-To-Rank

The training data in LTR consists of n query groups $D = \{\{x_{i,j}\}_{j=1}^{L_i}, \{y_{i,j}\}_{j=1}^{L_i}\}_{i=1}^{n}$. The i-th *query group* consists of L_i items (e.g. products) represented by feature vectors $\boldsymbol{x}_{i,j} \in \mathbb{R}^d$, and relevance labels $y_{i,j}$ which could be binary, ordinal, or real-valued measurements of relevance (Qin et al., 2021). The objective is to learn a function that, given a query group k, ranks the L_k items $\{x_{k,j}\}_{j=1}^{L_k}$ such that the items with highest relevance are ranked at the top. In this paper, we consider unsupervised pretraining, so we also have a larger unlabeled dataset D', which contains m query groups where $D' = \{\{x_{i,j}\}_{j=1}^{L_i}\}_{i=1}^{m}$ where $m \geq n$ and the query groups of D are a subset of those in D'. Most LTR algorithms formulate the problem as learning a function $f_\theta : \mathbb{R}^d \to \mathbb{R}$ that maps the feature vector associated with an item to a score, and then ranks the items by the scores.

To measure the quality of a ranking induced by our scoring function f_θ on the k-th query group, a commonly-used metric is NDCG:

$$\text{NDCG}(\pi_s, \{y_{k,j}\}_{j=1}^{L_k}) = \frac{\text{DCG}(\pi_s, \{y_{k,j}\}_{j=1}^{L_k})}{\text{DCG}(\pi^*, \{y_{k,j}\}_{j=1}^{L_k})}$$

where π_s $\pi_s : [L_k] \to [L_k]$ is a permutation over the L_k elements of the kth query group induced by the scoring function f_θ on $\{x_{k,j}\}_{j=1}^{L_k}$ while π^* is the ideal ranked list sorted by (real-valued relevance vector) $\boldsymbol{y}$, and discounted cumulative gain (DCG) is defined as $\text{DCG}(\pi, \{y_{k,j}\}_{j=1}^{L_k}) = \sum_{j=1}^{n} \frac{2^{y_{k,j}}-1}{\log_2(1+\pi(j))}$. Typically, a truncated version of NDCG is used that only considers the top-u ranked items, denoted as NDCG@u.

3. Unsupervised pretraining for ranking

We adapt two unsupervised pretraining approaches from the class of *contrastive learning* methods; two of the best-known variants are SimCLR (Chen et al., 2020a) and SimSiam (Chen and He, 2021).

Intuitively, SimCLR (Chen et al., 2020a) takes each data point $x_{i,j}$ and produces stochastically augmented versions $x_{i,j}^{(1)}$ and $x_{i,j}^{(2)}$ which are called a *positive pair*. Augmentations are designed to preserve the semantics of the data (e.g., rotating or blurring an image). SimCLR then trains a neural net to learn a common representation for two augmented views of the data (i.e., maximizing similarity between positive pairs), while minimizing similarity to negative pairs, which are augmentations of two semantically-different input samples. Details can be found in Appendix B.

SimSiam (Chen and He, 2021) is similar, except it does not use negative pairs. Instead, it only attempts to maximize similarity of representations of positive pairs. The full details are in Appendix B.

3.0.1. Adapting the methods to ranking

To adapt either of these methods to LTR, we must first choose a stochastic augmentation technique. In the image setting, augmentations for SimSiam and SimCLR include image flipping, cropping, and Gaussian blur (Chen and He, 2021; Chen et al., 2020a). In ranking, the choice of augmentation is less obvious; changing the values of features could change the entire nature of the data (which does not preserve semantics). In Section 4 we investigate several proposed augmentations, including randomly zeroing out feature values, randomly swapping feature values within a query group, and adding Gaussian noise.

Second, we should choose the unit of data to augment. In ranking, one can consider a single data point either as the full list of feature vectors in a query group or an individual feature vector in a query group. Viewing the entire query group as a data point accounts for interactions between items in the query group to get good representations–however, query groups are of variable length, which makes it challenging to adapt SimCLR and SimSiam to use query groups as the data unit. Hence, we use individual feature vectors within the query groups as the data points to augment.

4. Empirical Study

We study three main questions: **(1) What is the right augmentation strategy for unsupervised pretraining in ranking?** To understand this, we study a variety of augmentation strategies on SimSiam and SimCLR performed on ranking datasets. We find several augmentation strategies that can outperform non-pretrained methods on both full dataset metrics and in particular on outlier metrics. **(2) How should we finetune for ranking?** Typically in the unsupervised pretraining literature, one freezes the pretrained model and learns a linear model using the frozen pretrained representations on the labeled data. We find that this does not work in ranking, and instead either deep models have to be learned on top of the frozen representations or the entire network should be finetuned end-to-end. **(3) Where does the variance in test NDCG come from?** We find that, across multiple trials, deep methods (pretrained or otherwise) have significant NDCG variance. We explore a variety of training parameters and their effect on test NDCG variance. Finally, we combine the lessons learned in our empirical study to **recommend the most consistently performant training setting** including some ranking-specific design choices.

Dataset. We conduct our evaluation on three widely used public ranking datasets: MSLR-30k (Qin and Liu, 2013), Set1 from Yahoo (Chapelle and Chang, 2011), and Istella (Lucchese et al., 2016). Our experimental setup comes from PT-ranking (Yu, 2020). For all three datasets, we assume 0.1% of query groups in the training set of each dataset is labeled, while the rest is unlabeled. To evaluate our methods, we provide both the plain test NDCG@5 (measure of performance on the entire dataset) and the robust-NDCG@5 (measure of performance on queries with outlier feature values). We use the term *NDCG* to mean NDCG@5. We provide the precise methodology used to measure robust-NDCG in the following subsection.

Robust-NDCG. In interactive ML systems like search, performing well on outlier queries is particularly valuable as it empowers users to search for more outlier queries, which in turn allows the practitioner to collect more data on outliers and improve the model. To this end, we design a metric that evaluates NDCG only on outlier datasets.

When the outliers are not already known, we select outliers as follows: we generate a histogram with 100 bins for each feature across the *validation dataset*. For example, MSLR has 136 features, so we had 136 different histograms. For each histogram, we scan from left to right on the bins until we have encountered at least G empty bins in a row, and if there is less than 1% of the validation set above this bin, then all the feature values above this bin are considered outliers. We also repeat this process right to left. Any test query group with items that had outlier feature values were determined to be outlier query groups, and placed in the test outlier dataset. Robust-NDCG is then defined as the NDCG on this outlier dataset. Because different datasets have different sized typical gaps, we tune G for each dataset (MSLR, Yahoo, Istella) such the resulting percentage of outlier queries is as close to 1% of the test set as we can get. MSLR has $G = 5$, with 0.65% (40/6072) outlier queries, Yahoo has $G = 20$, with 1.4% (30/2147) outlier queries, and Istella has $G = 32$, with 0.46% (34/7202) outlier queries.

4.1. Choosing augmentation for unsupervised pretraining in ranking

We study three different choices of augmentation for ranking: “zeros”, “qg”, and “Gaussian”. As a running example, suppose the input to the augmentation module consists of four 5-dimensional vectors $[1, 2, 3, 4, 5]$, $[6, 7, 8, 9, 10]$; $[11, 12, 13, 14, 15]$, $[16, 17, 18, 19, 20]$ where the first two vectors represent items in query group A and the last two vectors represent items in query group B. In “zeros”, given an input, we zero out a random selection of features independently across samples. One possible output of applying the zeros augmentation to the example input is: $[1, 2, 0, 4, 0]$, $[6, 0, 8, 9, 10]$; $[0, 12, 0, 0, 15]$, $[16, 0, 18, 19, 20]$. In “qg”, given an input we randomly select features to substitute with values of the same column in the same query group. An example output of applying qg to the example input is: $[1, 2, 3, 4, 10]$, $[6, 2, 8, 9, 10]$; $[16, 12, 13, 14, 15]$, $[16, 12, 13, 19, 20]$. Finally, in the Gaussian augmentation we add zero mean Gaussian noise to all features and samples independently. For zeros and qg, we introduce a tuneable parameter c, which determines the probability that any given value is swapped out with a zero (in the case of zeros) or another value in the same column belonging to the same query group (in the case of qg). For the Gaussian augmentation we use unit Gaussian noise.

4.2. Experimental results

We compare SimSiam and SimCLR with various augmentation choices against two baselines that do not use unsupervised pretraining. The first is "GBDT", which is a gradient boosted decision tree implemented on lightgbm (Ke et al., 2017), trained on the labeled training data. The second is "No pretrain", which is an MLP trained only on the labeled training data. Training details are in Subsection C.2.

Test NDCG values are listed in Table 3. We bold the best two entries for each metric, and underline the worst entry for each metric. We observe the following takeaways:

Many augmentations outperform non-pretrained methods on all datasets. SimSiam qg with $c = 0.7$, SimCLR Gaussian, and SimCLR zeros $c = 0.2$ all outperform the non-pretrained models on every full dataset metric (sometimes significantly). SimSiam qg with $c = 0.7$ outperforms non-pretrained models on every metric, full or outlier test sets.

More augmentations outperform non-pretrained models on all robust metrics. We find that unsupervised pretraining increases robustness against real outlier data. Every SimSiam experiment (except for SimSiam zeros $c = 0.2$) has higher robust-NDCG values than non-pretrained methods across all datasets, while SimCLR achieved higher robust-NDCG values on Yahoo and Istella.

Variance can be high in neural ranking. The standard errors for non-pretrained and pretrained models' test NDCG were relatively high, suggesting some training instability.

Our outlier detection algorithm has shortcomings. Our outlier detection algorithm assumes that a small group of data far away from 99% of the data should be considered outliers, and will be harder to rank. However, in Istella, robust-NDCG values were higher than full dataset test-NDCG values for every method we tried. We found that in Istella there are features where the vast majority of the data has a value of zero (Figure 1). We hypothesize that zero represents a missing data value in the feature while the nonzero values have meaning, which makes the outliers our algorithm detects in Istella easier to rank than the "normal" values; hence, our outlier detection technique does not capture the samples that are hardest to rank.

4.3. How to finetune in ranking

Typically in unsupervised pretraining literature the preferred way to compare pretraining methods against each other is via *linear probing*; i.e., pretrain a model on unlabeled data to produce embeddings, freeze it, and learn a linear model using frozen embedding on the labeled data (Chen and He, 2021; Chen et al., 2020a; Grill et al., 2020; Chen et al., 2020b). An alternative choice is to add a linear layer on top of the embedder, but finetune the deep model end-to-end without freezing weights. A third alternative is to freeze the pretrained model and learn a model with possibly multiple layers using its frozen representations on the labeled data, which we call *non-linear probing*. We compare these three methods on our ranking problem. Training details in Subsection C.3.

Results. In Table 4 we detail our results. We bold the best entries for each metric, and underline the worst for each metric for both SimSiam and SimCLR. The takeaways:

Neither SimSiam nor SimCLR can perform well using linear probing. Both full dataset and robust metrics suffer greatly compared to the no-pretrain baseline when we have one finetuning layer.

End-to-end finetuning performs the best or close to the best. Overall, we suggest to use full finetuning as it generally works for both SimSiam and SimCLR.

4.4. Sources of variance in test NDCG results

In our earlier results, we found that the variance in the test NDCG results for our methods was high relative to the variance of GBDTs, which had a variance of zero on our datasets. In this section, we identify the sources of variance when utilizing pretraining methods in ranking. We do this by performing ablation studies on weight initialization, batch size, model architecture, the augmentation c parameter, and training algorithm with respect to their effects on test NDCG variance. Details are in Section C.4.

Overall, we find that there are two main sources of variance: (1) Finetuning variance comes from random initialization of added layers and finetuning data shuffling. These phenomena can be reduced by increasing the batch size of the finetuning phase and fixing the initialization of the additional finetuning layers to the identity. (2) Pretraining variance comes from augmentation, random initialization, and shuffling of pretraining data, which is mainly reduced by using more stable optimization algorithms and activation functions. We make the following observations from our experiments.

Finetuning is a source of variance, and fixing the initialization to the identity largely removes this variance. The evidence for this is summarized in Table 5. In summary, we find that if we fix the pretrained model and finetune it with default settings (random initialization, random batches, etc.) the variance in results is similar to if the pretrained model was not fixed. However, if we fix the finetuning layers' initialization to the identity, we can decrease the stderr of our results by an order of magnitude. And if in addition we

Table 1. The first row contains our results for GBDTs, the second is the no pretrain baseline, third is the best numbers achieved by SimSiam on all settings reported in Table 3, fourth is best numbers achieved by SimCLR-Rank (plus all the training recommendations in Subsection 4.5) on all settings reported in Table 3, and the final set of numbers is the result after utilizing the training recommendations in Section 4.5, which (1) obtains large performance improvements on MSLR30K and Istella full dataset NDCG, (2) achieves top-2 performance on every metric except for Yahoo's robust-NDCG. We report numbers as averages over 5 trials.

Method	MSLR30K	robust	Yahoo	robust	Istella	robust
GBDT	0.2799	0.1599	0.5342	0.4721	0.5450	0.5473
No pretrain	0.3464 ± 0.0083	0.2572	0.5752 ± 0.0048	0.4922	0.5387 ± 0.0056	0.6084
Best SimSiam (Table 3)	0.3600 ± 0.0032	**0.2953**	**0.6112 ± 0.0041**	**0.5224**	0.5512 ± 0.0065	0.6494
Best SimCLR (Table 3)	0.3561 ± 0.0054	0.2547	0.6109 ± 0.0020	0.5174	0.5690 ± 0.0102	0.659
SimCLR-Rank (recommended)	**0.3720 ± 0.0007**	0.2867	0.6109 ± 0.0020	0.5164	**0.5728 ± 0.0023**	**0.6719**

do full batch training, the stderr in our results completely disappears.

Reducing sources of variance in pretraining is not enough to get rid of pretraining variance. The evidence for this is summarized in Table 6. In summary, we find that in pretraining, we can remove the stochasticity from initialization, batch size, and decrease the augmentation level to $c = 0.01$ (for SimSiam zeros) and still have high stderr in results (even higher than what we had without all these interventions!). We cannot remove all sources of stochasticity in our pretraining because the pretraining relies upon stochastic augmentation, so we must try to make the pretraining process itself more robust against the noise introduced during the pretraining.

We can reduce the effect of augmentation variance on test NDCG results by properly choosing training and model architecture settings. The evidence is summarized in Table 6. While diagnosing sources of instability in the pretraining, we observed two behaviors of SimSiam: (1) output vectors during pretraining were often zero, which prevented the model from learning about the unlabeled samples, and (2) even under full batch training the pretraining loss could climb. To fix these two issues, we tried using ELU gating (Clevert et al., 2015) (as opposed to the default GeLU gating (Hendrycks and Gimpel, 2016)) to prevent too many values from being zeroed out, and used the SGD optimizer instead of Adam (Adam is known to potentially not converge (Reddi et al., 2019) and finds sharper minima (Keskar and Socher, 2017)). We find that these two modifications, along with the identity initialization and full batch training increased the stability of the results significantly, even if we set $c = 0.7$ with zeros augmentation. We conclude that model/training choices that do not directly introduce variance into the training process can have a large role in influencing the test NDCG variance.

4.5. Practical recommendations and their synthesis

Based on empirical observation and past literature, in this section, we first provide several recommendations for unsupervised pretraining for LTR. Notably, we modify the SimCLR loss (as **SimCLR-Rank**) for ranking datasets to reduce training time and memory cost. We synthesize these recommendations to produce a single pretraining and finetuning strategy which achieves strong results in all our datasets and metrics. We compare this "recommended setting" against the best SimSiam and SimCLR results from our augmentation study (Subsection 4.1) as well as against the non-pretrained methods. Results are in Table 1.

Recommendation: Now we provide several recommendations for pretraining and finetuning an LTR model.

1. In terms of overall metrics, finetuning and non-linear probing perform much better than linear probing for LTR (Table 4). In contrast, in the image domain, all strategies perform similarly, or under label scarcity linear probing outperforms full finetuning (Chen et al., 2020b). However, full finetuning is known to reduce robustness. Hence we recommend and adopt a procedure of first non-linear probing with three additional layers (100 epochs) and then full finetuning (200 epochs). This is similar to LP-FT (Kumar et al., 2022), except more layers are added.

2. Use SimCLR Gaussian pretraining. We observed that it performed consistently well across datasets (Table 3).

3. Use larger models for SimCLR pretraining. In image domain, SimCLR benefits from larger model size and batchesize (Chen et al., 2020b) as they are believed to digest more information from the unlabeled data. Therefore, we added 15 more layers and increased the batchsize. We increased the noise variance from 1 to 2 since the model has more layers and hence it has larger capacity to de-noise and learn.

4. Use a large projection head for SimCLR (Section B.1) and finetune from the first layer of this head to improve performance. We adopt this practice from Chen et al. (2020b).

5. Increase training stability to reduce the somewhat high variance we observed in test-NDCG across different runs. To remedy this, we added residual connections (He et al., 2016), used the tabular variant of ResNet (Gorishniy et al., 2021), and clipped the norm of the gradients to 2 and 1 for pretraining and finetuning, respectively.

6. Although we wanted to use larger models and batch sizes, we were significantly throttled by their slow training speed and high GPU memory usage. To remedy this, we introduce a new *ranking-aware* pretraining loss, **SimCLR-Rank**, which simplifies the SimCLR loss to reduce its GPU memory and time cost. Instead of contrasting the embedding of each item with the embeddings of all the other items in the batch (containing multiple query groups), SimCLR-Rank contrasts the item embedding with only the embeddings of other items in the same query group. We give the precise SimCLR-Rank loss in Section D.1 and compare it to SimCLR's loss.

Effectively, SimCLR-Rank reduces memory and time complexity from $O((NL)^2)$ to $O(NL^2)$ where N is the number of query groups per batch and L is the number of items per query group in that batch. SimCLR-Rank brought us substantial savings in runtime (20x) and memory. It allowed us to increase the number of model parameters by more than 2x and increase the batchsize by more than 10x (with more room to grow). We hypothesize that contrasting only within a query group is likely sufficient for the LTR setting, because in the downstream LTR task we need to only distinguish between items within query groups.

Results. Finally, in Table 1, we compare the empirical performance of the SimCLR-Rank model which incorporates all the above recommendations with the best SimCLR and SimSiam numbers from Table 3 on the three datasets. First, SimCLR-Rank with the same exact training choices across datasets, is able to obtain large performance gains on MSLR30K and Istella full dataset NDCGs. Second, consistently across datasets and metrics, SimCLR-Rank is the best or is amongst the top-2 best models for that metric (aside from Yahoo's robust-NDCG, where it is close). Further, SimCLR-Rank achieves this with significantly lower stderr when compared to our prior runs of SimCLR and SimSiam.

5. Conclusion

Through a comprehensive empirical study, we have found the training strategies and settings–including SimCLR-Rank–that are best able to make unsupervised pretraining with SimSiam and SimCLR on ranking data successful. We hope that our work will encourage adoption of unsupervised pretraining in the ranking domain and provide more understanding on the nuances of applying machine learning methods to the LTR problem.

References

Sebastian Bruch, Masrour Zoghi, Mike Bendersky, and Marc Najork. 2019. Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19)*. 1241–1244.

Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*. PMLR, 1–24.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020a. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. 2020b. Big self-supervised models are strong semi-supervised learners. *Advances in neural information processing systems* 33 (2020), 22243–22255.

Xinlei Chen and Kaiming He. 2021. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 15750–15758.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021), 18932–18943.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems* 33 (2020), 21271–21284.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. 2019. Using self-supervised learning can improve model robustness and uncertainty. *Advances in neural information processing systems* 32 (2019).

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. 2020. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678* (2020).

Alan Jeffares, Tennison Liu, Jonathan Crabbé, Fergus Imrie, and Mihaela van der Schaar. 2023. TANGOS: Regularizing Tabular Neural Networks through Gradient Orthogonalization and Specialization. *arXiv preprint arXiv:2303.05506* (2023).

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. *Advances in neural information processing systems* 30 (2017).

Nitish Shirish Keskar and Richard Socher. 2017. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628* (2017).

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. 2022. Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution. In *International Conference on Learning Representations*.

Sudarshan Dnyaneshwar Lamkhede and Christoph Kofler. 2021. Recommendations and Results Organization in Netflix Search. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 577–579.

Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. 2016. Post-learning optimization of tree ensembles for efficient ranking. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 949–952.

Kushal Majmundar, Sachin Goyal, Praneeth Netrapalli, and Prateek Jain. 2022. Met: Masked encoding for tabular data. *arXiv preprint arXiv:2206.08564* (2022).

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

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

Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Mike Bendersky, and Marc Najork. 2021. Are neural rankers still outperformed by gradient boosted decision trees? (2021).

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. 2019. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237* (2019).

Robin Swezey, Aditya Grover, Bruno Charron, and Stefano Ermon. 2021. Pirank: Scalable learning to rank via differentiable sorting. *Advances in Neural Information Processing Systems* 34 (2021), 21644–21654.

Talip Ucar, Ehsan Hajiramezanali, and Lindsay Edwards. 2021. Subtab: Subsetting features of tabular data for self-supervised representation learning. *Advances in Neural Information Processing Systems* 34 (2021), 18853–18865.

Shuo Yang, Sujay Sanghavi, Holakou Rahmanian, Jan Bakus, and Vishwanathan SVN. 2022. Toward Understanding Privileged Features Distillation in Learning-to-Rank. *Advances in Neural Information Processing Systems* 35 (2022), 26658–26670.

Hai-Tao Yu. 2020. PT-Ranking: A Benchmarking Platform for Neural Learning-to-Rank. arXiv:2008.13368 [cs.IR]

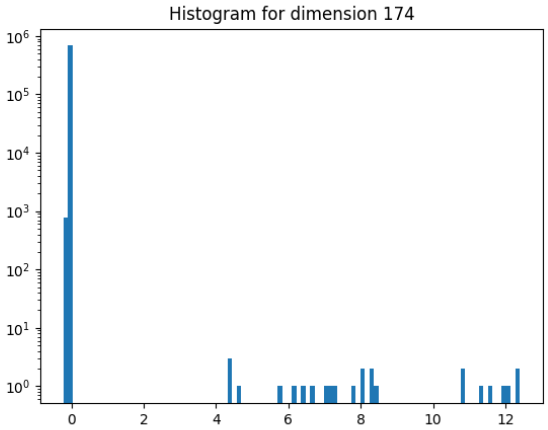

Figure 1. Histogram across all samples in Istella in feature 174. Note that nearly all the values are zero, while the vast minority of values with nonzero values have a lot of variation in them. We hypothesize that zero represents a missing data value while the nonzero values have a data value, which makes the outliers actually easier to rank than the "normal" values.

A. Related work

Classical LTR work. The current state-of-the-art for problems with tabular data is tree-based learners. In particular, the dominant model currently used in tabular LTR problems is gradient boosted decision trees (GBDTs) (Friedman, 2001). GBDT models, which perform well on tabular data, are adapted to the LTR setting via losses that are surrogates for ranking metrics like NDCG. Surrogate losses (including LambdaRank/RankNet (Burges, 2010) and PiRank (Swezey et al., 2021)) are needed because many important ranking metrics (like NDCG) are non-differentiable. The combination of tree-based learners and ranking losses has become the de-facto standard in ranking problems, and deep models have yet to outperform them convincingly (Qin et al., 2021).

Deep tabular models. Given the success of neural methods in many other domains, there have been many attempts to adapt deep models to the tabular domain. TANGOS introduced special tabular-specific regularization to try to improve deep models' performance (Jeffares et al., 2023). FT-transformer and TabTransformer were introduced as transformer-based approaches to tabular data (Gorishniy et al., 2021; Huang et al., 2020). All these models have failed to convincingly outperform tree-based methods based on their own evaluations.

Self-supervised learning. Self-supervised learning (SSL) has improved performance in settings where there is a significant source of unlabeled data like text (Devlin et al., 2018) and images (Chen et al., 2020a). In SSL, deep models are first pretrained on unlabeled data to learn useful representations for the data, and are then finetuned on labeled data. The core idea behind prominent SSL approaches like SimSiam and SimCLR is to carefully perturb input training samples, and train a representation that is consistent for transformations of the same sample. This provides robustness to natural perturbations and noise in data (Hendrycks et al., 2019). SSL has recently been applied to the tabular domain (Majmundar et al., 2022; Ucar et al., 2021) (MET, SubTab). Neither were evaluated in the ranking setting, and both were unable to convincingly and consistently outperform GBDTs in their experimental evaluations on real tabular data. Hence, it is unclear how to apply SSL to tabular LTR problems.

B. Background

B.1. SimCLR

We first summarize the original SimCLR (Chen et al., 2020a). First, for a data point $x_{i,j}$, we produce stochastically augmented versions $x_{i,j}^{(1)}$ and $x_{i,j}^{(2)}$ which are called a *positive pair*. Second, a base encoder $h(\cdot)$ and projection head $g(\cdot)$ map $x_{i,j}^{(1)}$ to $z_{i,j}^{(1)} = g(h(x_{i,j}^{(1)}))$ and $x_{i,j}^{(2)}$ to $z_{i,j}^{(2)} = g(h(x_{i,j}^{(2)}))$. Then we optimize the InfoNCE loss (Oord et al., 2018) to push $z_{i,j}^{(1)}$ and $z_{i,j}^{(2)}$ closer to each other in cosine similarity, and $z_{i,j}^{(1)}$ farther from other augmented data points (also in cosine

Table 2. Comparison of augmentations across datasets on test NDCG (higher is better). Averaged over 5 trials. We bold the best two values on each metric, and underline the worst value. GBDT is the GBDT baseline, "no pretrain" is the deep model baseline without pretraining. The GBDT baseline has no variance because the labeled training dataset is small. We observe two methods that outperform non-pretrained methods on all metrics: SimSiam qg with $c = 0.7$, and SimCLR zeros $c = 0.2$.

Method	MSLR30K	robust	Yahoo	robust	Istella	robust
GBDT	0.2799	0.1599	0.5342	0.4721	0.5450	0.5473
No pretrain	0.3464 ± 0.0083	0.2572	0.5752 ± 0.0048	0.4922	0.5387 ± 0.0056	0.6084
SimSiam zeros $c = 0.2$	0.3559 ± 0.0086	**0.2953**	$\mathbf{0.6140 \pm 0.0058}$	**0.5197**	0.5252 ± 0.0077	0.6052
SimSiam qg $c = 0.2$	0.3345 ± 0.0333	0.2615	0.6095 ± 0.0079	0.5145	0.5335 ± 0.0059	0.6204
SimSiam zeros $c = 0.7$	$\mathbf{0.3600 \pm 0.0032}$	0.2685	0.6079 ± 0.0080	**0.5224**	0.5390 ± 0.0105	0.6333
SimSiam qg $c = 0.7$	$\mathbf{0.3581 \pm 0.0022}$	**0.2815**	0.6008 ± 0.0073	0.5093	0.5512 ± 0.0065	0.6334
SimSiam Gaussian	0.3556 ± 0.0052	0.2744	$\mathbf{0.6112 \pm 0.0041}$	0.5117	0.5422 ± 0.0059	**0.6494**
SimCLR zeros $c = 0.2$	0.3533 ± 0.0066	0.2547	0.6019 ± 0.0078	0.5041	$\mathbf{0.5689 \pm 0.0102}$	0.6424
SimCLR qg $c = 0.2$	0.3428 ± 0.0053	0.2282	0.6044 ± 0.0050	0.5174	0.5501 ± 0.0020	0.6342
SimCLR zeros $c = 0.7$	0.3416 ± 0.0117	0.2294	0.5999 ± 0.0045	0.5144	0.5479 ± 0.0106	0.6232
SimCLR qg $c = 0.7$	0.3435 ± 0.0082	0.2410	0.6109 ± 0.0020	0.5147	0.5434 ± 0.01031	0.6315
SimCLR Gaussian	0.3561 ± 0.0054	0.2519	0.6063 ± 0.0027	0.5171	$\mathbf{0.5657 \pm 0.0074}$	**0.659**

Table 3. Comparison between SimCLR and SimCLR-rank.

Method	MSLR30K	robust	Yahoo	robust	Istella	robust
SimCLR Gaussian	0.3561 ± 0.0054	0.2519	0.6063 ± 0.0027	0.5171	0.5657 ± 0.0074	0.659
SimCLR-rank Gaussian	0.3534 ± 0.0059	0.2508	0.6052 ± 0.0027	0.5171	$\mathbf{0.5805 \pm 0.0022}$	0.6353

similarity) (Chen et al., 2020a). The main idea is to train the neural net to give a common representation to two augmented views of the data (maximize similarity between positive pairs), while not allowing the neural net to simply give the same representation to all inputs (minimize similarity to negative pairs).

B.2. SimSiam

SimSiam (Chen and He, 2021) similarly takes a data point $x_{i,j}$ and produces stochastically-augmented versions $x_{i,j}^{(1)}$ and $x_{i,j}^{(2)}$, which are called a *positive pair*. We pass the first sample of the pair through the base encoder $h(\cdot)$, projector $g(\cdot)$, and predictor $\text{pred}(\cdot)$, to get $p_{i,j}^{(1)} = \text{pred}(g(h(x_{i,j}^{(1)})))$; we pass the second sample through just the base encoder and projector to get $z_{i,j}^{(2)} = g(h(x_{i,j}^{(2)}))$. Then we maximize the cosine similarity of $p_{i,j}^{(1)}$ and $z_{i,j}^{(2)}$, while treating $z_{i,j}^{(2)}$ as a constant in the backpropagation. Unlike SimCLR, there are no "negative" pairs, i.e., the loss function does not try to push the representation of $z_{i,j}^{(1)}$ farther from other samples' augmentations.

C. Empirical Study

C.1. Datasets

For all three datasets, we assume 0.1% of query groups in the training set of each dataset is labeled, while the rest is unlabeled. To evaluate our methods, we provide both the plain test NDCG@5 (measure of performance on the entire dataset) and the robust-NDCG@5 (measure of performance on queries with outlier feature values). We use the term *NDCG* to mean NDCG@5. We provide the precise methodology used to measure robust-NDCG in the following subsection.

C.2. Details on augmentation study

Baselines. We compare against two baselines that do not use unsupervised pretraining. The first is "GBDT", which is a gradient boosted decision tree implemented on lightgbm (Ke et al., 2017), trained on the labeled training data. We grid

Table 4. Effect of changing the number of linear probing layers on pretrained models wrt to test NDCG and test robust-NDCG. "end-to-end" refers to finetuning the entire network. "Finetune strategy" refers to the number of layers appended to the end of the frozen pretrained model that are finetuned. Numbers are from single runs, except for "end-to-end" (where we are finetuning end-to-end, or full finetuning), which are an average over 5 runs. A few observations: (1) just one layer leads to unacceptable performance across all datasets across both methods (2) more layers helps both the SimSiam and SimCLR models, but helps the SimCLR model much more (3) more layers increases the performance on robust metrics (4) with enough layers appended, SimCLR with frozen pretrained representations is competitive with fully finetuned SimCLR. We bold the best entries for each metric, and underline the worst for each metric for both SimSiam and SimCLR.

Method	Finetune Strategy	MSLR30K	Yahoo	Istella
No pretrain	-	0.3464	0.5753	0.5387
SimSiam	+1 layer (linear probing)	<u>0.2208</u>	<u>0.4234</u>	<u>0.2812</u>
SimSiam	+4 layer (non-linear probing)	0.2362	0.5097	0.3573
SimSiam	+7 layer (non-linear probing)	0.2364	0.4687	0.3472
SimSiam	end-to-end	**0.3559**	**0.6140**	**0.5512**
SimCLR	+1 layer (linear probing)	<u>0.2930</u>	<u>0.4713</u>	<u>0.3117</u>
SimCLR	+4 layer (non-linear probing)	0.3331	0.6009	0.5385
SimCLR	+7 layer (non-linear probing)	0.3587	0.5994	0.5496
SimCLR	+10 layer (non-linear probing)	0.3513	0.5973	0.5477
SimCLR	end-to-end	**0.3561**	**0.6063**	**0.5658**

searched on the following parameters: num_leaves [16, 31, 96], n_estimators [10, 31, 96, 200], min_child_samples [10, 20, 60, 200], learning_rate [0.1, 0.01], used early stopping based on validation accuracy and reported the best test accuracy. Second is "No pretrain", which is a five-layer MLP (multi-layer perceptron) trained only on the labeled training data, and using the default parameters from PT-ranking. The loss function used for both baselines is LambdaRank (Burges, 2010).

Training details. For the pretrained models, we select an augmentation and value of c and then pretrain with either SimSiam or SimCLR for 300 epochs on the entire training dataset. For SimSiam, we use a sample (as opposed to query group) batch size of roughly 200000, while for SimCLR we have a batch size of roughly 5000. In general, it has been found that larger batch sizes benefit both SimSiam and SimCLR (Chen and He, 2021), so we try to set it as high as possible for each method. In the pretraining phase, we train with a learning rate of 5e-4 while in the finetuning phase we train with a learning rate of 1e-5. We do full finetuning, i.e. the entire network is trained during the finetuning phase. The optimizer we use is Adam with default Pytorch parameters. We finetune the entire network, which is a five-layer MLP like the "No pretrain" baseline, and also use the LambdaRank loss (Burges, 2010).

C.3. Details on finetuning in ranking

In this section we study different finetuning methods and how they impact final test NDCG.

Background. Typically in unsupervised pretraining literature the preferred way to compare pretraining methods against each other is via linear probing. In linear probing, one pretrains a model, freezes it, and learns a linear model using its frozen representations on the labeled data (Chen and He, 2021; Chen et al., 2020a; Grill et al., 2020; Chen et al., 2020b). One alternative choice to this is finetuning the deep model end to end without freezing after adding one or more layers on top of it (we call this full finetuning). A second alternative is to freeze the pretrained model and learn a model with possibly multiple layers using its frozen representations on the labeled data, which we call probing. We will compare these three methods on our ranking problem.

Training details. Aside from "no pretrain" and entries with "end-to-end" (meaning that we do full finetuning where we update all weights during training), we take a particular checkpoint of a pretrained model, add one or more linear layers (number is specified by Layers column), and finetune only those layers with a learning rate of 1e-5. All are single runs, except for "no pretrain" and when "end-to-end", which are over five trials and copied over from Table 3. For SimSiam, we use a pretrained checkpoint that was pretrained using the qg augmentation at $c = 0.7$. For SimCLR, we use a pretrained checkpoint that was pretrained using the Gaussian augmentation.

Table 5. Ablation study on sources of variance during finetuning. We find that much of the variability can be attributed to the random initialization of finetuning layers. Details in Subsubsection C.4.1.

Method	MSLR30K	Yahoo	Istella
Baseline	0.3561 ± 0.0054	0.6063 ± 0.0027	0.5657 ± 0.0074
Default finetuning settings	0.3487 ± 0.0057	0.6039 ± 0.0026	0.5718 ± 0.0018
Identity Finetune Init	0.3512 ± 0.0005	0.6039 ± 0.0005	0.5550 ± 0.0004
Identity Finetune Init + Full batch	0.3549	0.6007	0.5340

Table 6. Ablation study on sources of variance in pretraining. We find that optimizer (e.g. Adam vs SGD) and model choices (like activation functions) have a big impact on the stability of results. Details in Subsubsection C.4.2.

Method	MSLR30K	Yahoo	Istella
Baseline	0.3561 ± 0.0054	0.6063 ± 0.0027	0.5657 ± 0.0074
Identity Pretrain Init + Full batch	0.3308 ± 0.0178	0.5976 ± 0.0027	0.5608 ± 0.0010
Identity Pretrain Init + Full batch + $c = 0.01$	0.3273 ± 0.0196	0.5964 ± 0.0053	0.5477 ± 0.0118
Identity Pretrain Init + Full batch + $c = 0.7$ + SGD opt + ELU	0.2770 ± 0.0010	0.6187 ± 0.0015	0.4101 ± 0.0016

C.4. Details on variance study

C.4.1. Finetuning variance

In Table 5 we find that the main source of variance in finetuning comes from the random initialization of the finetuning layers. Methodology: we fix a pretrained model that was pretrained using SimCLR with Gaussian augmentation, and perform an ablation study on various training choices in the finetuning, where the choices are: {Default finetuning settings, Identity Finetune Init, Identity Finetune Init + Full batch}. In the first setting, we simply finetune using the protocol we had used before (random initialization of finetuning layers, batched and shuffled training). In the second, we add five layers on top of the pretrained model and initialize all matrices to the identity and the bias to zero. In the third, we both use the fixed initializations above and also perform full gradient descent in the finetuning phase. We compare all of these against the baseline, which is a model that is pretrained with SimCLR Gaussian and finetuned with default finetuning settings. All results are averages over five trials, and the metric we use is NDCG.

C.4.2. Pretraining variance

In Table 6 we perform an ablation study on possible sources of pretraining variance when we zero out the finetuning variance (by using full batch training in finetuning and fixing the finetuning initializations as in Table 5). In this table we find that variance coming from augmentation in the pretraining is unavoidable, but we can dampen the effect of it by: (1) removing other sources of variance like initialization and batch size, (2) adopting training strategies that are less noisy. The baseline is a pretrained model that was pretrained using SimSiam qg 0.7 using the default pretraining and finetuning settings from Section 4.1. We have four different pretraining settings that we perform an ablation study over: {Identity Pretrain Init + Full batch, Identity Pretrain Init + Full batch + $c = 0.01$, Identity Pretrain Init + Full batch + $c = 0.7$ + SGD opt + ELU }. From the first two experiments, we find that removing initialization variance, batch size variance, and decreasing augmentation variance as much as possible was insufficient to reduce pretraining variance. In the last experiment, we use SGD and ELU activation in the pretraining (as opposed to Adam and GeLU/ReLU), which reduced the variance significantly. However, average test NDCG suffered greatly for two out of three datasets (all except Yahoo Set1).

D. Synthesis Details

D.1. SimCLR-Rank vs SimCLR

To keep this section self-contained, we will repeat the exposition on SimCLR in Section B.1.

First, for a data point $x_{i,j}$ (i-th query group, j-th item in the query group), we produce stochastically augmented versions $x_{i,j}^{(1)}$ and $x_{i,j}^{(2)}$ which are called a *positive pair*. Second, a base encoder $h(\cdot)$ and projection head $g(\cdot)$ map $x_{i,j}^{(1)}$ to $z_{i,j}^{(1)} = g(h(x_{i,j}^{(1)}))$

and $x_{i,j}^{(2)}$ to $z_{i,j}^{(2)} = g(h(x_{i,j}^{(2)}))$. Then we optimize the InfoNCE loss (Oord et al., 2018) to push $z_{i,j}^{(1)}$ and $z_{i,j}^{(2)}$ closer to each other in cosine similarity, and $z_{i,j}^{(1)}$ farther from other augmented data points (also in cosine similarity) (Chen et al., 2020a). Precisely, the loss for SimCLR looks like this for each augmented datapoint $x_{i,j}$:

$$\begin{aligned}\ell_{i,j} = &-\log \frac{\exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{i,j}^{(2)})/\tau)}{\sum_{q=1}^{B}\sum_{k=1}^{L_q} \mathbf{1}[i \neq q \text{ or } k \neq j][\exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{q,k}^{(1)})/\tau) + \exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{q,k}^{(2)})/\tau)]} \\ &+ -\log \frac{\exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{i,j}^{(2)})/\tau)}{\sum_{q=1}^{B}\sum_{k=1}^{L_q} \mathbf{1}[i \neq q \text{ or } k \neq j][\exp(\mathrm{sim}(z_{i,j}^{(2)}, z_{q,k}^{(1)})/\tau) + \exp(\mathrm{sim}(z_{i,j}^{(2)}, z_{q,k}^{(2)})/\tau)]}\end{aligned}$$

Where B is the number of query groups in the batch, τ is a temperature parameter, and sim$(\cdot)$ and the final loss is averaged across all the available (i, j) in the batch. For SimCLR-Rank, the loss is replaced with

$$\begin{aligned}\ell_{i,j} = &-\log \frac{\exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{i,j}^{(2)})/\tau)}{\sum_{k=1}^{L_i} \mathbf{1}[k \neq j][\exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{i,k}^{(1)})/\tau) + \exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{i,k}^{(2)})/\tau)]} \\ &+ -\log \frac{\exp(\mathrm{sim}(z_{i,j}^{(1)}, z_{i,j}^{(2)})/\tau)}{\sum_{k=1}^{L_i} \mathbf{1}[k \neq j][\exp(\mathrm{sim}(z_{i,j}^{(2)}, z_{i,k}^{(1)})/\tau) + \exp(\mathrm{sim}(z_{i,j}^{(2)}, z_{i,k}^{(2)})/\tau)]}\end{aligned}$$

Where the final loss is averaged across all the available (i, j) in the batch. Note the difference in the denominator, which now is summed only over the items within the query group as opposed to the entire batch.