The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

# A Batch Learning Framework for Scalable Personalized Ranking

**Kuan Liu, Prem Natarajan**
Information Sciences Institute & Department of Computer Science
University of Southern California

**Abstract**

In designing personalized ranking algorithms, it is desirable to encourage a high precision at the top of the ranked list. Existing methods either seek a smooth convex surrogate for a non-smooth ranking metric or directly modify updating procedures to encourage top accuracy. In this work we point out that these methods do not scale well in a large-scale setting, and this is partly due to the inaccurate pointwise or pairwise rank estimation. We propose a new framework for personalized ranking. It uses batch-based rank estimators and smooth rank-sensitive loss functions. This new batch learning framework leads to more stable and accurate rank approximations compared to previous work. Moreover, it enables explicit use of parallel computation to speed up training. We conduct empirical evaluations on three item recommendation tasks, and our method shows a consistent accuracy improvement over current state-of-the-art methods. Additionally, we observe time efficiency advantages when data scale increases.

## Introduction

The task of personalized ranking is to provide each user with a ranked list of items that he might prefer. It has received considerable attention in academic research (Hu, Koren, and Volinsky 2008; Rendle et al. 2009; He et al. 2016), and algorithms developed are applied in various applications in e-commerce (Linden, Smith, and York 2003), social networks (Chen et al. 2009), location (Liu et al. 2014), etc. However, personalized ranking remains a very challenging task: 1) The learning objectives of ranking models are hard to directly optimize. For example, the quality of the model output is commonly evaluated by ranking measures such as NDCG, MAP, and MRR, which are position-dependent (or rank-dependent) and non-smooth. It makes gradient-based optimization infeasible and also computationally expensive. 2) The size of an item set that a ranking task uses can be very large. It is not uncommon to see an online recommender system with millions of items. As a consequence, it increases the difficulty of capturing user preferences over the entire set of items. It also makes it harder to compute or estimate the rank of a particular item.

Traditional approaches model user preferences with *rank-independent* algorithms. Pairwise algorithms convert the learning task into many binary classification problems and optimize the *average classification accuracy*. For example, BPR (Rendle et al. 2009) maximizes the probability of correct prediction of each pairwise item comparison. MMMF (Srebro, Rennie, and Jaakkola 2005) minimizes a margin loss for each pair in a matrix factorization setting. Listwise algorithms such as those recently explored by (Hidasi et al. 2015; Covington, Adams, and Sargin 2016) treat the problem as a multi-class classification and use cross-entropy as the loss function.

Despite the simplicity and wide application, these rank-independent methods are not satisfactory because the quality of results from a ranking system is highly position-dependent. A high accuracy at the top of a list is more important than that at a low position on the list. However, the average accuracy targeted by the pairwise algorithms discussed above places equal weights at all the positions. This mismatch therefore leads to under-exploitation in the prediction accuracy at the top part. Listwise algorithms, on the other hand, do make an attempt to push items to the top using a classification scheme. However, its classification criterion also does not match well with ranking.

Position-dependent approaches are explored to address the above limitations. One critical question is how to approximate item ranks to perform rank-dependent training. TFMAP (Shi et al. 2012a) and ClifMF (Shi et al. 2012b) approximate an item rank purely based on the model score of this item, i.e., a **pointwise estimate**. Particularly, it models the reciprocal rank of an item with a sigmoid transformed from the score returned by the model. TFMAP then optimizes a smoothed modification of MAP, while ClifMF optimizes MRR. This pointwise estimation is simple, but it is only loosely connected to the true ranks. The estimation becomes even more unreliable as the itemset size increases.

An arguably more direct approach is to optimize the ranks of relevant items returned by the model to encourage top accuracy. It requires the computation or estimation of item ranks and modification of the updating strategy. This idea is explored in traditional learning to rank methods LambdaNet (Burges et al. 2005), LambdaRank (Burges, Ragno, and Le 2007), etc., where the learning rate is adjusted based on item ranks. In personalized ranking, WARP (Weston, Bengio, and Usunier 2010) propose to use a sampling procedure to estimate item ranks. It repeatedly samples negative items until it finds one that has a higher score. Then the number of sampling trials is used to estimate item ranks. This stochastic **pairwise estimation** is intuitive. WARP is also found to be more competitive than BPR (Hong, Doumith, and Davison 2013). A more recent matrix factorization model LambdaFM (Yuan et al. 2016) adopts the same rank estimation. However, this pairwise rank estimator becomes very noisy and unstable as the itemset size increases (as we will demonstrate). It takes a large number of sampling iterations for each estimation. Moreover, its intrinsic online learning fashion prevents full utilization of available parallel computation (e.g., GPUs), making it hard to train large or flexible models which rely on the parallel computation.

The limitations of these approaches largely come from the stochastic pairwise estimation component. As a comparison, training with batch or mini-batch together with parallel computation has recently offered opportunities to tackle scalability challenges. (Hidasi et al. 2015) use a sampled classification setting to train a RNNs-based recommender system. (Covington, Adams, and Sargin 2016) deploy a two-stage classification system in YouTube video recommendation. Parallel computation (e.g., GPUs) are extensively used to accelerate training and support flexible models.

In this work we propose a novel framework to do personalized ranking. Our aim is to have better rank approximations and advocate the top accuracy in large-scale settings. The contributions of this work are:

- We propose rank estimations that are based on batch computation. They are shown to be more accurate and stable in approximating the true item rank.
- We propose smooth loss functions that are "rank-sensitive." This advocates top accuracy by optimizing the loss function values. Being differentiable, the functions are easily implemented and integrated with back-propagation updates.
- Based on batch computation, the algorithms explicitly utilize parallel computation to speed up training. They lend themselves to flexible models which rely on more extensive computation.
- Our experiments on three item recommendation tasks show consistent accuracy improvements over state-of-the-art algorithms. They also show time efficiency advantages when data scale increases.

The remainder of the paper is organized as follows: We first introduce notations and preliminary methods. We next detail our new methods, followed by discussions on related work. Experimental results are then presented. We conclude with discussions and future work.

## Notations

In this paper we will use the letter $x$ for users and the letter $y$ for items. We use unbolded letters to denote single elements of users or items and bolded letters to denote a set of items. Particularly, a single user and a single item are, respectively, denoted by $x$ and $y$, $\mathbf{Y}$ denotes the entire item set. $\mathbf{y}_x$ denotes the positive items of user $x$—that is, the subset of items that are interacted by user $x$. $\bar{\mathbf{y}}_x \equiv \mathbf{Y} \setminus \mathbf{y}_x$ is the irrelevant item set of user $x$. We omit subscript x when there is no ambiguity. $\mathbf{S} = \{(x, y)\}$ is the set of observations. The indicator function is denoted by $\mathbf{I}$. f denotes a model (or model parameters). $f_y(x)$ denotes the model score for user $x$ and item $y$. For example, in a matrix factorization model, $f_y(x)$ is computed by the inner product of latent factors of $x$ and $y$. Given a model $f$, user $x$, and its positive items $\mathbf{y}$, the rank of an item $y$ is defined as

$$r_y \equiv rank_y(f, x, \mathbf{y}) = \sum_{y' \in \bar{\mathbf{y}}} \mathbf{I}[f_y(x) \leq f_{y'}(x)], \qquad (1)$$

where we use the same definition as in (Usunier, Buffoni, and Gallinari 2009) and ignore the order within positive items. The indicator function is sometimes approximated by a continuous margin loss $|1 - f_y(x) + f_{y'}(x)|_+$ where $|t|_+ \equiv max(t, 0) \quad \forall t \in \mathbb{R}$.

## Position-dependent personalized ranking

Position-dependent algorithms take the ranks of predicted items into account in the model training. A practical challenge is how to estimate item ranks efficiently. As seen in (1), the ranks depend on the model parameters and are dynamically changing. The definition is non-smooth in model parameters due to the indicator function. The computation of ranks involves comparisons with all the irrelevant items, which can be costly.

Existing position-dependent algorithms address the challenge by different rank approximation methods. They can be categorized into pointwise and pairwise approximations. We describe their approaches in the following.

### Pointwise rank approximation

Item ranks are approximated in TFMAP (Shi et al. 2012a) and ClifMF (Shi et al. 2012b) using a pointwise approach. Particularly,

$$r_y \approx rank_y^{point}(f, x) = 1/\sigma(f_y(x)), \qquad (2)$$

where $\sigma(z) = 1/(1 + e^{-z}), \forall z \in \mathbb{R}$. The approximated rank $r_y$ is then plugged into an evaluation metric MAP (as in TFMAP) and MRR (as in ClifMF) to make the objective smooth. The algorithms then use gradient-based methods for optimization.

In (2), $rank_y^{point}$ is close to 1 when model score $f_y(x)$ is high and becomes large when $f_y(x)$ is low. This connection between model scores and item ranks is intuitive, and implicitly encourages a good accuracy at the top. However, $rank_y^{point}$ is very loosely connected to rank definition in (1). In practice, it does not capture the non-smooth characteristics of ranks. For example, small differences in model scores can lead to dramatic rank differences when the item set is large.

### Pairwise rank approximation

An alternative approach used by WARP (Weston, Bengio, and Usunier 2010), LambdaMF (Yuan et al. 2016) estimates item ranks based on comparisons between a pair of items.

The critical component is an **iterative sampling approximation procedure**: given a user x and a positive item y, keep sampling a negative item $y' \in \bar{\mathbf{y}}$ uniformly with replacement until the condition $1 + f_{y'}(x) < f_y(x)$ is violated. With the sampling procedure it estimates item ranks by

$$r_y \approx rank_y^{pair}(f, x, \mathbf{y}) = \lfloor \frac{|\bar{\mathbf{y}}| - 1}{N} \rfloor \tag{3}$$

where N is the number of sampling trials to find the first violating example and $\lfloor z \rfloor$ takes the maximum integer that is no greater than $z$.

The intuition behind this estimation is that the number of trials of sampling follows a geometric distribution. Suppose the item's true rank is $r_y$, the probability of having a violating item is $p = r_y/(|\bar{\mathbf{y}}| - 1)$. The expected number of trials is $\mathbb{E}(N) = 1/p = (|\bar{\mathbf{y}}| - 1)/r_y$.

To promote top accuracy, the estimated item ranks are used to modify updating procedures. For example, in WARP, they are plugged in a loss function defined as,

$$L^{owa}(x, y) = \Phi^{owa}[r_y] = \sum_{j=1}^{r_y} \alpha_j \quad \alpha_1 \geq \alpha_2 \geq .. \geq 0 \tag{4}$$

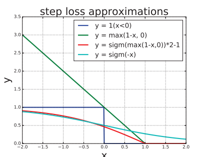
where $\alpha_j \ j = 1, 2, ..$ are predefined non-increasing scalars. The function $\Phi^{owa}$ is derived from ordered weighted average (OWA) of classification loss (Yager 1988) in (Usunier, Buffoni, and Gallinari 2009). It defines a penalty incurred by placing $r_y$ irrelevant elements before a relevant one. Choosing equal $\alpha_j$ means optimizing the mean rank, while choosing $\alpha_{j>1} = 0, \alpha_1 = 1$ means optimizing the proportion of top-ranked correct items. With strictly decreasing $\alpha$s, it optimizes the top part of a ranked list.

## Approach

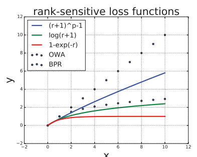
We begin by pointing out several limitations of approaches based on pairwise rank estimations.

First, the rank estimator in (3) is not only biased but also has a large variance. Expectation of the estimator in (3) for $p$ of a geometric distribution is approximately $p(1 + \sum_{k=2}^{N} \frac{1}{k}(1-p)^{k-1}) > p$. In a ranking example where $p = r/N$ (N population size), it overestimates the rank seriously when r is small. Moreover, we will demonstrate later that the estimator has high estimation variances. We believe this poor estimation may lead to training inefficiency. Second, it can take a large number of sampling iterations before finding a violating item in each iteration. This is especially the case after the beginning stage of training. This results in low frequency of model updating. In practice, prolonged training time is observed. Finally, the *intrinsic sequential learning fashion* based on pairwise estimation prevents full utilization of available parallel computation (e.g., GPUs). This makes it hard to train large or flexible models which highly rely on the parallel computation.

We address the limitations by combining the ideas of batch computation and rank-dependent training loss. Particularly, we propose batch rank approximations and generalize (4) to smooth rank-sensitive loss functions. The resulting



(a) Indicator approximations.



(b) Rank-sensitive loss functions.

Figure 1: Illustrations of rank approximations and smooth rank-sensitive loss functions. 1a shows different approximations of indicator functions. 1b shows smooth loss functions used to generalize the loss (4).

algorithm gives more accurate rank approximations and allows back-propagation updates.

## Batch rank approximations

In order to have a stable and accurate rank approximation that leads to efficient algorithms. We exploit the idea of batch training which has been recently actively explored or revisited in areas such as model design (Covington, Adams, and Sargin 2016) and optimization (Chen et al. 2016).

To begin, we define *margin rank* (mr) as the following:

$$rank_y^{mr}(f, x, \mathbf{y}) = \sum_{y' \in \bar{\mathbf{y}}} |1 - f_y(x) + f_{y'}(x)|_+, \tag{5}$$

where the margin loss is used to replace the indicator function in (1), and the target item $y$ is compared to a batch of negative items $\bar{\mathbf{y}}$. As illustrated in Figure 1a, the margin loss (green curve) is a smooth convex upper bound of the original step loss function (or indicator function). *Margin rank* sums up the margin errors and characterizes the overall violation of irrelevant items.

*Margin rank* could be sensitive to "bad" items that have significantly higher scores than the target item. As seen in Eq. (5) or Figure 1a, such a bad item contributes much more than one in rank estimation. To suppress that effect, we add a sigmoid transformation, i.e.,

$$rank_y^{smr}(f, x, \mathbf{y}) = \sum_{y' \in \bar{\mathbf{y}}} 2 * \sigma(|1 - f_y(x) + f_{y'}(x)|_+) + 1, \tag{6}$$

where $\sigma(z) = 1/(1 + e^{-z}), \forall z \in \mathbb{R}$. We call this *suppressed margin rank* (smr). Additionally, we study a smoother version without margin formulation, i.e., $rank_y^{sr}(f, x, \mathbf{y}) = \sum_{y' \in \bar{\mathbf{y}}} \sigma(f_{y'}(x) - f_y(x))$. Therefore, our rank approximations can be written as

$$rank_y^{batch}(f, x, \mathbf{y}) = \sum_{y' \in \bar{\mathbf{y}}} \tilde{r}(x, y, y'), \tag{7}$$

where $\tilde{r}(x, y, y')$ takes one of the following three forms:

- $\tilde{r}(x, y, y') = |1 - f_y(x) + f_{y'}(x)|_+$
- $\tilde{r}(x, y, y') = 2 * \sigma(|1 - f_y(x) + f_{y'}(x)|_+) - 1$
- $\tilde{r}(x, y, y') = \sigma(f_{y'}(x) - f_y(x))$

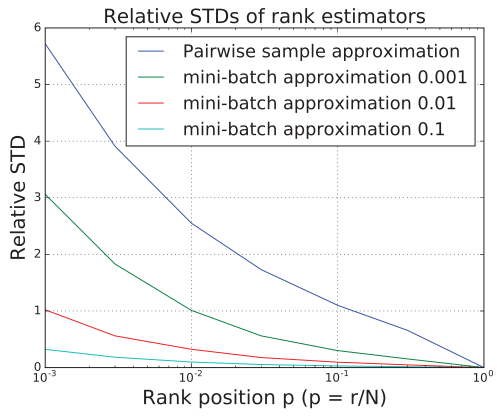


Figure 2: Standard deviations (relative values) of two types of rank estimators at different item ranks. Simulation is done with item set size N=100,000. 'online' uses estimator (3) and 'sampled-batch q' uses (8) where $q = |\mathbf{Z}|/|\mathbf{Y}|$.

Note that the rank approximations in (7) are computed in a batch manner: model scores between a user and a batch of items are first computed in parallel; $\tilde{r}$ are then computed accordingly and summed up. This batch computation allows model parameters to be updated more frequently. The parallel computation can speed up model training.

The full batch treatment may become infeasible or inefficient when the item set gets overly large. A mini-batch approach is used for that case. Instead of computing the rank based on the full set $\mathbf{Y}$ as in (7), the mini-batch version algorithm samples $\mathbf{Z}$, a subset of $\mathbf{Y}$ randomly (without replacement) and computes

$$rank_y^{mb}(f, x, \mathbf{y}) = \frac{|\mathbf{Y}|}{|\mathbf{Z}|} \sum_{y' \in \mathbf{Z}} \tilde{r}(x, y, y') \mathbf{I}(y' \in \bar{\mathbf{y}}). \quad (8)$$

Although a sampling step is involved, we argue below that (8) does not lead to large variances as the sampling in pairwise approaches do. First, (8) is an unbiased estimator of (suppressed) margin rank. Second, the sampling schemes in the two approaches are different. To have a better idea, we conduct a simulation of two types of sampling and plot standard deviations. Figure 2 shows that (8) has much smaller variances than (3) as long as $|\mathbf{Z}|/|\mathbf{Y}|$ is not too small.

## Smooth rank-sensitive loss functions

The rank-based loss function (4) operates on item ranks and provides a mechanism to advocate top accuracy. However, its input has to be an integer. It is also non-smooth. Consequently, it is not applicable to our batch rank estimators and does not support gradient-based optimization. In the following, we generalize (4) to smooth rank-sensitive loss functions that similarly advocate top accuracy.

We first observe that a loss function $\ell$ would encourage top accuracy when the following three conditions are satisfied—we then call it rank sensitive (rs):

1. $\ell$ is a smooth function of the rank r.
2. $\ell$ is increasing, i.e., $\ell' > 0$.
3. $\ell$ is concave, i.e., $\ell'' < 0$.

The second condition indicates that the loss increases when the rank of a target item increases. Thus, given a single item, minimizing the objective would try to push the item to the top. The third condition means the increase is fast when rank is small and slow when rank is large. So it is more sensitive to small ranking items. Given more than one item, an algorithm that minimizes this objective would prioritize on those items with small estimated rank values.

Based on the observation, we study several types of functions $\ell^{rs}$ that satisfy these conditions: polynomial functions, logarithmic functions, and exponential functions, i.e.,

- $\ell_1^{rs}(r) = (1+r)^p, \quad 0 < p < 1$
- $\ell_2^{rs}(r) = \log(r+1)$
- $\ell_3^{rs}(r) = 1 - \lambda^{-r}, \quad \lambda > 1$

It follows standard calculus to verify that these functions satisfy the above conditions. Thus, they all incur (smoothly) weighed loss based on estimated rank values and advocate top accuracy. We plot part of these functions in Figure 1b and compare them to BPR and OWA (with $\alpha s = 1, 1/2, 1/3, \ldots$). BPR is equivalent to a linear function which places a uniformly increasing penalty on the estimated rank. Polynomial and logarithmic functions have a diminishing return when the estimated rank increases and is unbounded. Exponential functions are bounded by 1, and the penalty on high rank values is quickly saturated.

## Algorithm

An algorithm can be then formulated as minimizing an objective based on the (mini-)batch approximated rank values and rank-sensitive loss functions. It sums over all pairs of observed user-item activity. Particularly,

$$L = \sum_{(x,y) \in \mathbf{S}} \ell^{rs}(r_y^{mb}(f, x, \mathbf{y})) + \phi(f), \quad (9)$$

where $r_y^{mb}$ is given by (8) and $\ell^{rs}$ takes $\ell_i^{rs}, i = 1, 2, 3$. $\phi(f)$ is a model regularization term.

Gradient-based methods are used to solve the optimization problem. The gradient with respect to the model can be written as

$$\frac{\partial L}{\partial f} = \sum \ell'(r) \times \frac{\partial r}{\partial f}, \quad (10)$$

where $\ell'(r)$ takes the form $p(1+r)^{p-1}$ (or $1/(r+1)$, or $\lambda^{-r}$) and we ignore the regularization term for the moment. We call the framework defined in (9) **Batch-Approximated-Rank-Sensitive loss** (BARS). The details are described in Algorithm 1.

Note that in Algorithm 1 computation is conducted in a batch manner. Particularly, it computes model scores between a mini-batch of users ($\mathbf{x}$) and sampled items ($\mathbf{Z}$) in parallel. In every step it updates parameters of all the users in $\mathbf{x}$ and items in $\mathbf{Z}$.

*Comparisons to Lambda-based methods.* Lambda-based methods such as LambdaNet and LambdaRank use an updating strategy in the following form

$$\delta f_{ij} = \lambda_{ij} * |\Delta NDCG_{ij}|, \quad (11)$$

**Algorithm 1:**

```
Input: Training data S; mini-batch size $m$; Sample
       rate $q$; a learning rate $\eta$.
Output: The model parameters f.
initialize parameters of model f randomly;
while Objective (9) is not converged do
    sample a mini-batch of observations $\{(x, y)_i\}_{i=1}^{m}$;
    sample item subset Z from Y, $q = |\mathbf{Z}|/|\mathbf{Y}|$;
    compute approximated ranks by (8);
    update model f parameters:
        $f = f - \eta * \partial \ell / \partial f$ based on (10);
end
```

where $\lambda_{ij}$ is a regular gradient term and $|\Delta NDCG_{ij}|$ is the NDCG difference of the ranking results for a specific user if the positions (ranks) of items i, j get switched, and acts as a scaling term that is motivated to directly optimize the ranking measure.

Compare (10) and (11): $\ell'(r)$ replaces $|\Delta NDCG_{ij}|$ and functions in a similar role. In our cases, $\ell'(r)$ is decreasing in r. Thus it gives a higher incentive to fix errors in low-ranking items. However, instead of directly computing the NDCG difference which can be computationally expensive in large-scale settings, the proposed algorithm first approximates the rank and then computes the scaling value through derivative of the loss function.

## Related work

**Top accuracy in traditional ranking.** Top accuracy is a classical challenge for ranking problems. One typical approach is to develop smooth surrogates of the non-smooth ranking metric.(Weimer et al. 2008) use structure learning and propose smooth convex upper bounds of a ranking metric. (Taylor et al. 2008) develop a smooth approximation based on connections between score distribution and rank distribution. Similarly, in a kernel SVM setting, (Agarwal 2011) propose a formulation based on infinity norm. (Boyd et al. 2012) generalize the notion of top k to top $\tau$-quantile and optimize a convex surrogate of the corresponding ranking loss.

Alternatively, (Burges et al. 2005) start with the average precision loss and modify model updating steps to promote top accuracy. Similar ideas are developed in (Burges, Ragno, and Le 2007; Wu et al. 2010). (Burges, Ragno, and Le 2007) write down the gradient directly rather than deriving it from a loss. (Wu et al. 2010) work on a boosted tree formulation.

**Personalized ranking.** Traditional work on personalized ranking does not necessarily focus on top accuracy. (Hu, Koren, and Volinsky 2008) first study the task and convert it as a regression problem. (Rendle et al. 2009) introduce ranking-based optimization and optimize a criterion similar to AUC. (He et al. 2016) improves matrix factorization by giving missing items non-uniform weights and devising an ALS based solver.

To promote top accuracy and deal with large-scale settings, (Shi et al. 2012a) develop rank approximation based on model score and propose a smooth approximation of MAP. (Shi et al. 2012b) adopt the same idea and target MRR.

Pairwise algorithms (Weston, Bengio, and Usunier 2010; Yuan et al. 2016) are then proposed to estimate ranks through sampling methods. (Weston, Bengio, and Usunier 2010) update the model based on an operator ordered weighted average. (Yuan et al. 2016) use a similar idea as in (Burges et al. 2005).

| Data | $\|U\|$ | $\|I\|$ | $\|S_{train}\|$ | $\|S_{test}\|$ |
|---|---|---|---|---|
| ML-20m | 138,493 | 27,278 | 7,899,901 | 2,039,972 |
| Yelp | 1,029,433 | 144,073 | 1,917,218 | 213,460 |
| XING | 1,500,000 | 327,002 | 2,338,766 | 484,237 |

Table 1: Dataset statistics. U: users; I: items; S: interactions.

## Experiments

In this section we conduct experiments on three large-scale real-world datasets to verify the effectiveness of the proposed methods.

### Experimental setup

**Dataset** We validate our approach on three public datasets from different domains: 1) movie recommendations; 2) business reviews at Yelp; 3) job recommendations from XING.[1] We describe the datasets in detail below.

**MovieLens-20m** The dataset has anonymous ratings made by MovieLens users.[2] We transform the data into binary indicating whether a user rated a movie above 4.0. We discard users with less than 10 movie ratings and use 70%/30% train/test splitting. Attributes include movie genres and movie title text.

**Yelp** dataset comes from Yelp Challenge.[3] We work on recommendations related to which business a user might want to review. Following the *online protocol* in (He et al. 2016), we sort all interactions in chronological order, take the last 10% for testing and take the rest for training. Business items have attributes including *city*, *state*, *categories*, *hours*, and *attributes* (e.g., "valet parking," "good for kids").

**XING** contains about 12 weeks of interaction data between users and items on XING. Train/test splitting follows the RecSys Challenge 2016 (Abel et al. 2016) setup where the last two weeks of interactions for a set of 150,000 *target users* are used as test data. Rich attributes are associated with data like career levels, disciplines, locations, job descriptions etc. Our task is to recommend to users a list of job posts with which they are likely to interact.

We report dataset statistics in Table 1.

**Methods** We study multiple algorithms under our learning framework and compare them to various baseline methods. Particularly, we study the following algorithms:

- POP. A naive baseline model that recommends items in terms of their popularity.

[1]www.xing.com

[2]www.movielens.org

[3]https://www.yelp.com/dataset_challenge. Downloaded Feb 17.

| Datasets | ML-20m | | | Yelp | | | XING | | |
|---|---|---|---|---|---|---|---|---|---|
| Metrics | P@5 | R@30 | NDCG@30 | P@5 | R@30 | NDCG@30 | P@5 | R@30 | NDCG@30 |
| POP | 6.2 | 10.0 | 8.5 | 0.3 | 0.9 | 0.5 | 0.5 | 2.7 | 1.3 |
| BPR | 6.1 | 10.2 | 8.3 | 0.1 | 0.4 | 0.2 | 0.3 | 2.2 | 0.9 |
| b-BPR | 9.3 | 14.3 | 12.9 | 0.9 | 3.4 | 1.9 | 1.3 | 9.2 | 4.2 |
| WARP | *10.1* | 13.3 | 13.5 | 1.3 | 4.3 | 2.5 | 2.6 | 11.6 | 6.7 |
| CE | 9.6 | 14.3 | 13.2 | *1.4* | 4.5 | 2.6 | 2.5 | *12.3* | 6.5 |
| SR-log | 9.9 | 14.5 | *13.6* | *1.4* | *5.2* | *2.9* | *2.8* | *12.3* | *6.9* |
| MR-poly | **10.2** | **14.8** | **13.9** | **1.5** | *5.2* | *2.9* | *2.8* | **12.5** | *6.9* |
| MR-log | **10.2** | *14.6* | **13.9** | **1.5** | 5.1 | *2.9* | **2.9** | **12.5** | **7.1** |
| SMR-log | **10.2** | *14.6* | **13.9** | **1.5** | **5.4** | **3.0** | **2.9** | **12.5** | **7.1** |

Table 2: Recommendation accuracy comparisons (in %). Results are averaged over 5 experiments with different random seeds. Best and second best numbers are in bold and italic, respectively.

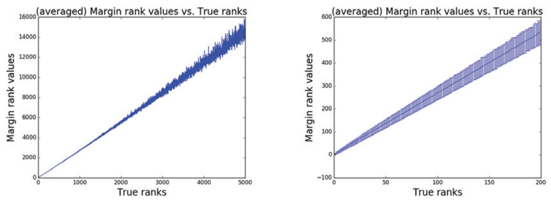
- BPR (Rendle et al. 2009). BPR optimizes AUC and is a widely used baseline.
- b-BPR. A batch version of BPR. It uses the same logistic loss but updates a target item and a batch of negative items every step.
- WARP (Weston, Bengio, and Usunier 2010; Hong, Doumith, and Davison 2013). A state-of-the-art pairwise personalized ranking method.
- CE. Cross entropy loss is recently used for item recommendation in (Hidasi et al. 2015; Covington, Adams, and Sargin 2016).
- SR-log. The proposed algorithm with the smoothed rank approximation without margin formulation (sr) and logarithmic function (log).
- MR-poly. The proposed algorithm with the margin rank approximation (mr) and polynomial function (poly).
- MR-log. The proposed algorithm with the margin rank approximation (mr) and logarithmic function (log).
- SMR-log. The proposed algorithm with the suppressed margin rank approximation (smr) and logarithmic function (log).

BPR and WARP are implemented by LIGHTFM (Kula 2015). We implemented the other algorithms.

We apply the algorithms to hybrid matrix factorization (Shmueli et al. 2012), a factorization model that represents users and items as linear combinations of their attribute embedding. Therefore, model parameters $f$ include factors of users, items, and their attributes.

Early stopping is used on a development dataset split from training for all models. Hyper-parameter model size is tuned in $\{10, 16, 32, 48, 64\}$; learning rate is tuned in $\{0.5, 1, 5, 10\}$; when applicable, dropout rate is 0.5. Batch-based approaches are implemented based on Tensorflow 1.2 on a single GPU (NVIDIA Tesla P100) ). LIGHTFM runs on Cython with a 5-core CPU (Intel Xeon 3.30GHz).

**Metrics** We assess the quality of recommendation results by comparing a model's recommendations to ground truth interactions, and report *Precision* (P), *Recall* (R) and *Normalized Discounted Cumulative Gain* (NDCG) scores. We report scores *after removing historical items from each user's recommendation list* on Yelp and ML-20m datasets because users seldom re-interact with items in these scenarios (Yelp reviews/movie ratings). This improves performance for all models but does not change relative comparison results.



(a) 0-50000. (b) 0-200.

Figure 3: Approximated rank values compared to the true rank values. 3b is a zoomed-in version with error bars.

## Results

**Quality of rank approximations** We first study how well the proposed methods approximate the true item ranks. To do that, we run one epoch of training on the XING dataset and compute the values in (5) and the true item rank. We plot the value of (5) as a function of the true rank in Figure 3.

Figure 3a shows in a very large range (0-50000) that the estimator in (5) is linearly aligned with the true item rank, especially when true item ranks are small—note that those regions are what we most care about. We further zoom into the top region (0-200) and plot error bars in addition to function mean values. In Figure 3b, we see limited variances. For example, the relative standard deviation is smaller than 0.1, which indicates stable rank approximation. As a comparison, the simulation in Figure 2 suggests that stochastic pairwise estimation should lead to a relative standard deviation much more than 6.

**Recommendation accuracy** Recommendation scores are reported in Table 2. We have the following observations.

First, vanilla pairwise BPR performs poorly due to a large itemset size. In contrast, batch version b-BPR bypasses the

| **Datasets** | ML-20m | Yelp | XING |
|---|---|---|---|
| Density | 2.6e-3 | 1.4e-5 | 5.8e-6 |
| # of param. | 4.6M | 9.3M | 12.1M |
| # of Attr. | 11 | 19 | 33 |
| $\|I\|$ | 27K | 144K | 327K |
| complexity | small | medium | large |

Table 3: Dataset/model complexity comparisons.

difficulty of sampling negative items and updates model parameters smoothly, achieving decent accuracies.

Second, WARP and CE outperform b-BPR. Both methods target more than averaged precision. Compared with each other, CE penalizes more on a correct item ranking behind incorrect items, thus showing consistently better performances in recall.

Third, the proposed methods consistently outperform WARP and CE. Compared to CE, the improvements suggest the effectiveness of the rank-sensitive loss, which works better than the classification loss. Compared to WARP, we attribute the improvements to better rank approximations and possibly other factors like smoother parameter updates.

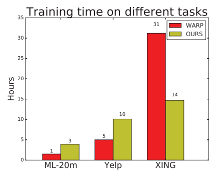
Finally, we compare the different variants of the proposed methods. SR-log underperforms, and it suggests the benefit of margin formulation. Polynomial loss functions have similar results compared to logarithmic functions, but require a bit tuning in the hyper-parameter $p$. Suppress margin rank (SMR) performs a bit better than MR—probably due to its better rank approximation.

**Time efficiency** We study the time efficiency of the proposed methods and compare them to the pairwise algorithm implementations. Note that we are *not* just interested in the comparisons of the absolute numbers because they involve multiple factors. Rather, we focus on the **trend** of how time efficiency changes when data scale increases.

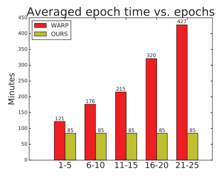
We characterize the dataset complexity in Table 3 by the density (computed by $\frac{\#.observations}{\#.users \times \#.items}$), the number of total parameters, the average number of attributes per user-item pair, and the itemset size. From Table 3, ML-20m has the densest observations, the smallest number of total parameters and attributes per user-item, and the smallest itemset size. Thus we call it "small" in complexity. Conversely, we call XING "large" in complexity. Yelp is between the two and is called "medium."

Two results are reported in Figure 4. Figure 4a shows across different datasets the converging time needed to reach the best models from the two systems: WARP and BARS (ours). WARP takes a shorter time in both "small" and "medium", but its running time increases very fast; BARS has a slower increase in the training time and wins at "large."

Figure 4b depicts the averaged epoch training time of the two systems. BARS has a constant epoch time. In contrast, WARP keeps increasing the training time per epoch. This is expected because when the model becomes better, it takes a lot more sampling iterations every step.



(a) Across datasets.



(b) Across time.

Figure 4: Training time comparisons between WARP and BARS. Fig. 4a plots how training time changes across datasets with different scales; Fig. 4b plots how epoch time changes as the training progresses.

| q | ML-20m | | Yelp | | XING | |
|---|---|---|---|---|---|---|
| | obj | NDCG | obj | NDCG | obj | NDCG |
| 1.0 | 6.49 | 16.0 | 7.94 | 3.0 | 6.42 | 9.9 |
| 0.1 | 6.47 | 15.9 | 7.87 | 3.0 | 6.40 | 10.0 |
| 0.05 | 6.47 | 15.9 | 7.90 | 2.9 | 6.42 | 9.8 |

Table 4: Comparisons of objective values (obj) and recommendation accuracies (NDCG) on development set among full batch and sampled batch algorithms. $q = |\mathbf{Z}|/|\mathbf{Y}|$, $q = 1.0$ means full batch.

**Robustness to mini-batch size** The full batch algorithm is used in the above experiments. We are also interested in seeing how it performs with a sampled batch loss. In Table 4 we report loss values and NDCG@30 scores on development split, and compare them to full batch versions. With the sampling proportion 0.1 or 0.05, the sampled version algorithm gives almost identical results as the full batch version on all datasets. This suggests the robustness of the algorithm to mini-batch size.

## Conclusion

In this work we address the personalized ranking task and propose a learning framework that exploits the ideas of batch training and rank-dependent loss functions. The proposed methods allow more accurate rank approximations and empirically give competitive results.

In designing the framework, we purposely tailored our approach to the use of parallel computation and support of back-propagation updates. This readily lends itself to flexible models such as deep feedforward networks, recurrent neural nets, etc. In the future, we are interested in exploring the algorithm in the training of deep neural networks.

## References

Abel, F.; Benczúr, A.; Kohlsdorf, D.; Larson, M.; and Pálovics, R. 2016. Recsys challenge 2016: Job recommendations. *Proceedings of the 2016 International ACM Recommender Systems.*

Agarwal, S. 2011. The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, 839–850. SIAM.

Boyd, S.; Cortes, C.; Mohri, M.; and Radovanovic, A. 2012. Accuracy at the top. In *Advances in neural information processing systems*, 953–961.

Burges, C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; and Hullender, G. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, 89–96. ACM.

Burges, C. J.; Ragno, R.; and Le, Q. V. 2007. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, 193–200.

Chen, J.; Geyer, W.; Dugan, C.; Muller, M.; and Guy, I. 2009. Make new friends, but keep the old: recommending people on social networking sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 201–210. ACM.

Chen, J.; Monga, R.; Bengio, S.; and Jozefowicz, R. 2016. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*.

Covington, P.; Adams, J.; and Sargin, E. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 191–198. ACM.

He, X.; Zhang, H.; Kan, M.-Y.; and Chua, T.-S. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 549–558. ACM.

Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; and Tikk, D. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

Hong, L.; Doumith, A. S.; and Davison, B. D. 2013. Co-factorization machines: modeling user interests and predicting individual decisions in twitter. In *Proceedings of the sixth ACM international conference on Web search and data mining*, 557–566. ACM.

Hu, Y.; Koren, Y.; and Volinsky, C. 2008. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, 263–272. Ieee.

Kula, M. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. *arXiv preprint arXiv:1507.08439*.

Linden, G.; Smith, B.; and York, J. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7(1):76–80.

Liu, Y.; Wei, W.; Sun, A.; and Miao, C. 2014. Exploiting geographical neighborhood characteristics for location recommendation. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 739–748. ACM.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, 452–461. AUAI Press.

Shi, Y.; Karatzoglou, A.; Baltrunas, L.; Larson, M.; Hanjalic, A.; and Oliver, N. 2012a. Tfmap: optimizing map for top-n context-aware recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, 155–164. ACM.

Shi, Y.; Karatzoglou, A.; Baltrunas, L.; Larson, M.; Oliver, N.; and Hanjalic, A. 2012b. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, 139–146. ACM.

Shmueli, E.; Kagian, A.; Koren, Y.; and Lempel, R. 2012. Care to comment?: recommendations for commenting on news stories. In *Proceedings of the 21st international conference on World Wide Web*, 429–438. ACM.

Srebro, N.; Rennie, J.; and Jaakkola, T. S. 2005. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, 1329–1336.

Taylor, M.; Guiver, J.; Robertson, S.; and Minka, T. 2008. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, 77–86. ACM.

Usunier, N.; Buffoni, D.; and Gallinari, P. 2009. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th annual international conference on machine learning*, 1057–1064. ACM.

Weimer, M.; Karatzoglou, A.; Le, Q. V.; and Smola, A. J. 2008. Cofi rank-maximum margin matrix factorization for collaborative ranking. In *Advances in neural information processing systems*, 1593–1600.

Weston, J.; Bengio, S.; and Usunier, N. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning* 81(1):21–35.

Wu, Q.; Burges, C. J.; Svore, K. M.; and Gao, J. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13(3):254–270.

Yager, R. R. 1988. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on systems, Man, and Cybernetics* 18(1):183–190.

Yuan, F.; Guo, G.; Jose, J. M.; Chen, L.; Yu, H.; and Zhang, W. 2016. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 227–236. ACM.