# Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System

Jiaxi Tang
School of Computing Science
Simon Fraser University
British Columbia, Canada
jiaxit@sfu.ca

Ke Wang
School of Computing Science
Simon Fraser University
British Columbia, Canada
wangk@cs.sfu.ca

## ABSTRACT

We propose a novel way to train ranking models, such as recommender systems, that are both effective and efficient. Knowledge distillation (KD) was shown to be successful in image recognition to achieve both effectiveness and efficiency. We propose a KD technique for learning to rank problems, called *ranking distillation (RD)*. Specifically, we train a smaller student model to learn to rank documents/items from both the training data and the supervision of a larger teacher model. The student model achieves a similar ranking performance to that of the large teacher model, but its smaller model size makes the online inference more efficient. RD is flexible because it is orthogonal to the choices of ranking models for the teacher and student. We address the challenges of RD for ranking problems. The experiments on public data sets and state-of-the-art recommendation models showed that RD achieves its design purposes: the student model learnt with RD has a model size less than half of the teacher model while achieving a ranking performance similar to the teacher model and much better than the student model learnt without RD.

## CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**; **Recommender systems**; **Retrieval efficiency**;

## KEYWORDS

Recommender System; Learning to Rank; Knowledge Transfer; Model Compression

**ACM Reference Format:**
Jiaxi Tang and Ke Wang. 2018. Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System. In *Proceedings of The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220021

## 1 INTRODUCTION

In recent years, information retrieval (IR) systems become a core technology of many large companies, such as web page retrieval for Google and Yahoo; personalized item retrieval *a.k.a* recommender systems for Amazon and Netflix. The core in such systems is a ranking model for computing the relevance score of each ($q$, $d$) pair for future use, where $q$ is the query (*e.g.,* keywords for web page retrieval and user profile for recommender systems) and $d$ is a document (*e.g.,* a web page or item). The *effectiveness* of IR systems largely depends on how well the ranking model performs, whereas the *efficiency* determines how fast the systems will respond to user queries, *a.k.a* online inferences.

Since the 2009 Netflix Prize competition, it is increasingly realized that a simple linear model with few parameters cannot model the complex query-document (user-item) interaction properly, and latent factor models with numerous parameters are shown to have better effectiveness and good representation power [21]. Recently, with the great impact of neural networks on computer vision [17, 22] and natural language processing [19, 25], a new branch of IR using neural networks has shown strong performances. As neural networks have incredible power to automatically capture features, recent works use neural networks to capture semantic representations for both queries and documents, relieving the manual feature engineering work required by other approaches. Several successful ranking models with neural networks have been investigated [14, 33, 35, 37]. However, the size of such models (in terms of the number of model parameters) increases by an order of magnitude or more than previous methods. While such models have better ranking performance by capturing more query-document interactions, they incur a larger latency at online inference phase when responding to user requests due to the larger model size.

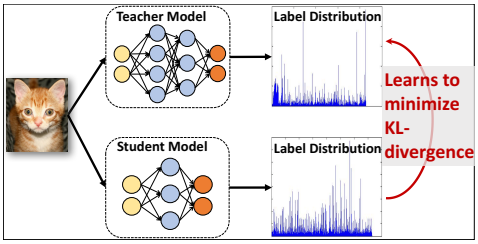
Balancing effectiveness and efficiency has been a line of recent research [23, 34, 39, 41, 42]. Discrete hashing techniques [39, 40, 42] and binary coding of model parameters are suggested to speed up the calculation of the relevance score for a given ($q$,$d$) pair. Other works focus on database-related methods, such as pruning and indexing to speed-up retrieval of related items [23, 34], using fast models for candidate generation and applying time-consuming models to the candidates for online inferences [9, 24]. These methods either lose much of effectiveness, due to the introduced model constraints, or cannot be easily extended to other models in most cases, due to the model-dependency nature.

### 1.1 Knowledge Distillation

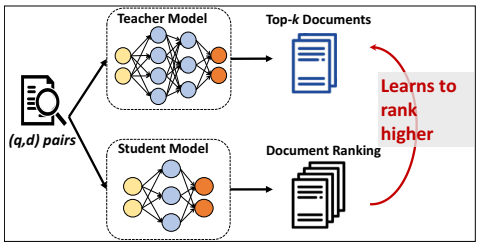
*Knowledge distillation* (KD), a.k.a., knowledge transfer, is a model-independent strategy for generating a compact model for better inference efficiency while retaining the model effectiveness [1, 3, 15]. The idea of KD is shown in Figure 1a for image recognition. During the offline training phase, a larger *teacher model* is first trained


*KDD '18, August 19–23, 2018, London, United Kingdom*

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220021




**(a) Knowledge Distillation for classification**




**(b) Ranking Distillation for ranking problems**

**Figure 1: (a) Knowledge Distillation: given an input, student model learns to minimize the KL Divergence of its label distribution and teacher model's. (b) Ranking Distillation: given an query, student model learns to give higher rank for it's teacher model's top-$K$ ranking of documents.**

from the training set, and a smaller *student model* is then trained by minimizing two deviations: the deviation from the training set's ground-truth label distribution, and the deviation from the label distribution generated by the teacher model. Then the student model is used for making online inferences. Intuitively, the larger teacher model helps capture more information of the label distribution (for example, outputting a high probability for "tiger" images for a "cat" image query due to correlation), which is used as an additional supervision to the training of the student model. The student model trained with KD has an effectiveness comparable to that of the teacher model [1, 15, 20] and can make more efficient online inference due to its small model size.

Despite of this breakthrough in image recognition, it is not straightforward to apply KD to ranking models and ranking problems (*e.g.*, recommendation). First, the existing KD is designed for classification problems, not for ranking problems. In ranking problems, the focus is on predicting the relative order of documents or items, instead of predicting a label or class as in classification. Second, KD requires computing the label distribution of documents for each query using both teacher and student models, which is feasible for image classification where there are a small number of labels, for example, no more than 1000 for the ImageNet data set; however, for ranking and recommendation problems, the total number of documents or items could be several orders of magnitudes larger, say millions, and computing the distribution for all documents or items for each training instance makes little sense, especially only the highly ranked documents or items near the top of the ranking will matter. We also want to point out that, for context sensitive recommendation, such as sequential recommendation, the items to be recommended usually depend on the user behaviors prior to the recommendation point (e.g., what she has viewed or purchased), and the set of contexts of a user is only known at the recommendation time. This feature requires recommender system to be strictly responsive and makes online inference efficiency particularly important.

## 1.2 Contributions

In this work, we study knowledge distillation for the learning to rank problem that is the core in recommender systems and many other IR systems. Our objective is achieving the ranking performance of a large model with the online inference efficiency of a small model. In particular, by fusing the idea of knowledge transfer and learning to rank, we propose a technique called *ranking distillation* (RD) to learn a compact ranking model that remains effective. The idea is shown in Figure 1b where a small student model is trained to learn to rank from two sources of information, *i.e.*, the training set and the top-$K$ documents for each query generated by a large well-trained teacher ranking model. With the large model size, the teacher model captures more ranking patterns from the training set and provides top-$K$ ranked unlabeled documents as an extra training data for the student model. This makes RD differ from KD, as teacher model in KD only generate additional labels on existing data, while RD generate additional training data and labels from unlabeled data set. The student model benefits from the extra training data generated from the teacher, in addition to the data from usual training set, thus, inherits the strong ranking performance of the teacher, but is more efficient for online inferences thanks to its small model size.

We will examine several key issues of RD, *i.e.*, the problem formulation, the representation of teacher's supervision, and the balance between the trust on the data from the training set and the data generated by the teacher model, and present our solutions. Extensive experiments on recommendation problems and real-world datasets show that the student model achieves a similar or better ranking performance compared to the teacher model while using less than half of model parameters. While the design goal of RD is retaining the teacher's effectiveness (while achieving the student's online inference efficiency), RD exceeds this expectation that the student sometime has a even better ranking performance than the teacher. Similar to KD, RD is orthogonal to the choices of student and teacher models by treating them as black-boxes. To our knowledge, this is the first attempt to adopt the idea of knowledge distillation to large-scale ranking problems.

In the rest of the paper, we introduce background materials in Section 2, propose ranking distillation for ranking problems in Section 3, present experimental studies in Section 4, and discuss related work in Section 5. We finally conclude with a summary of this work and in Section 6.

# 2 BACKGROUNDS

We first review the learning to rank problem, then revisit the issues of effectiveness and efficiency in the problem, which serves to motivate our ranking distillation. Without loss of generality, we use the IR terms "query" $q$ and "document" $d$ in our discussion, but

these terms can be replaced with "user profile" and "item" when applied to recommender systems.

## 2.1 Ranking from scratch

The learning to rank problem can be summarized as follows: Given a set of queries $\mathcal{Q}=\{q_1,\cdots,q_{|\mathcal{Q}|}\}$ and a set of documents $\mathcal{D}=\{d_1,\cdots,d_{|\mathcal{D}|}\}$, we want to retrieve documents that are most relevant to a certain query. The degree of relevance for a query-document pair $(q,d)$ is determined by a relevance score. Sometimes, for a single $(q,d)$ pair, a relevance score $y_d^{(q)}$ is labeled by human (or statistical results) as ground-truth, but the number of labeled $(q,d)$ pairs is much smaller compared to the pairs with unknown labels. Such labels can be binary (*i.e.,* relevant/non-relevant) or ordinal (*i.e.,* very relevant/relevant/non-relevant). In order to rank documents for future queries with unknown relevance scores, we need a ranking model to predict their relevance scores. A ranking model $M(q,d;\boldsymbol{\theta})=\hat{y}_d^{(q)}$ is defined by a set of model parameters $\boldsymbol{\theta}$ and computes a relevance score $\hat{y}_d^{(q)}$ given the query $q$ and document $d$. The model predicted document ranking is supervised by the human-labeled ground truth ranking. The optimal model parameter set $\boldsymbol{\theta}^*$ is obtained by minimizing a ranking-based loss function:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{q\in\mathcal{Q}} \mathcal{L}^R(\boldsymbol{y}^{(q)}, \hat{\boldsymbol{y}}^{(q)}). \tag{1}$$

For simplicity, we focus on a single query $q$ and omit the superscripts related to queries (*i.e.,* $y_d^{(q)}$ will becomes $y_d$).

The ranking-based loss could be categorized as point-wise, pair-wise, and list-wise. Since the first two are more widely adopted, we don't discuss list-wise loss in this work. The point-wise loss is widely used when relevance labels are binary [14, 33]. One typical point-wise loss is taking the negative logarithmic of the likelihood function:

$$\begin{aligned}\mathcal{L}^R(\boldsymbol{y},\hat{\boldsymbol{y}}) = -(&\sum_{d\in\boldsymbol{y}_{d+}} \log(P(rel=1|\hat{y}_d)) \\ &+ \sum_{d\in\boldsymbol{y}_{d-}} \log(1-P(rel=1|\hat{y}_d))),\end{aligned} \tag{2}$$

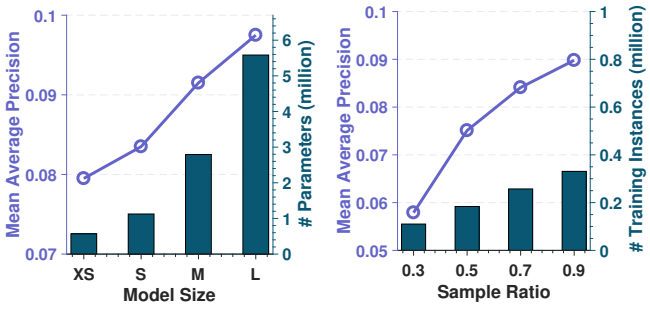
where $\boldsymbol{y}_{d+}$ and $\boldsymbol{y}_{d-}$ are the sets of relevant and non-relevant documents, respectively. We could use the *logistic* function $\sigma(x)=1/(1+e^{-x})$ and $P(rel=1|\hat{y}_d)=\sigma(\hat{y}_d)$ to transform a real-valued relevance score to the probability of a document being relevant ($rel=1$). For ordinal relevance labels, pair-wise loss better models the partial order information:

$$\mathcal{L}^R(\boldsymbol{y},\hat{\boldsymbol{y}}) = -\sum_{d_i,d_j\in C} \log(P(d_i \succ d_j|\hat{y}_i,\hat{y}_j)), \tag{3}$$

where $C$ is the set of document pairs $\{(d_i,d_j): y_i \succ y_j\}$ and the probability $P(d_i \succ d_j)$ can be modeled using the *logistic* function $P(d_i \succ d_j|y_i,y_j)=\sigma(y_i-y_j)$.

## 2.2 Rethinking Effectiveness and Efficiency

We consider ranking models with latent factors or neural networks (*a.k.a* neural ranking models) instead of traditional models (*e.g.,* SVM, tree-based models) for the following reasons. First, these models are well-studied recently for its capability to capture features from a latent space and are shown to be highly effective; indeed, neural ranking models are powerful for capturing rich semantics for queries and documents, which eliminates the tedious and ad-hoc feature extraction and engineering normally required in traditional models. Second, these models usually require many parameters and suffer from efficiency issue when making online inferences. Third, traditional models like SVM usually has convex guarantees and are trained through convex optimization. The objectives of latent factor models and neural networks are usually non-convex [8, 18], which means that their training processes are more challenging and need more attentions.




(a) MAP *vs.* model size (b) MAP *vs.* training instances

**Figure 2: Two ways of boosting mean average precision (MAP) on Gowalla data for recommendation. (a) shows that a larger model size in number of parameters, indicated by the bars, leads to a higher MAP. (b) shows that a larger sample size of training instances leads to a higher MAP.**

The goal of ranking models is predicting the rank of documents as accurately as possible near the top positions, through learning from human-labeled ground-truth document ranking. Typically, there are two ways to make a ranking model perform better at top positions: (1) By having a large model size, as long as it doesn't overfit the data, the model could better capture complex query-document interaction patterns and has more predictive capability. Figure 2a shows that, when a ranking model has more parameters, it acquires more flexibility to fit the data and has a higher MAP, where the mean average precision (MAP) is more sensitive to the precision at top positions. (2) By having more training data, side information, human-defined rules *etc.*, the model can be trained with more guidance and has less variance in gradients [15]. Figure 2b shows that, when more training instances are sampled from the underlying data distribution, a ranking model could achieve a better performance. However, each method has its limitations: method (1) surrenders efficiency for effectiveness whereas method (2) requires additional informative data, which is not always available or is expensive to obtain in practice.

# 3 RANKING DISTILLATION

In this section, we propose *ranking distillation* (RD) to address the dual goals of effectiveness and efficiency for ranking problems. To address the efficiency of online inference, we use a smaller ranking model so that we can rank documents for a given query more efficiently. To address the effectiveness issue without requiring more

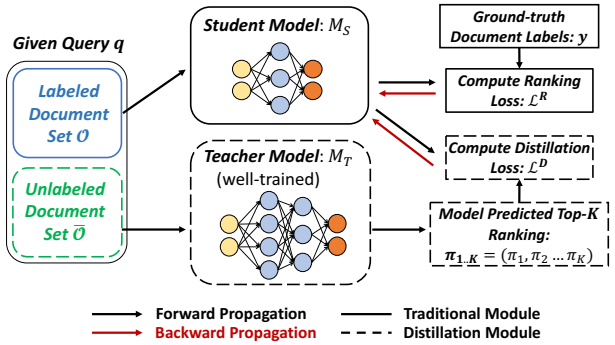



**Figure 3: The learning paradigm with ranking distillation. We first train a teacher model and let it predict a top-$K$ ranked list of unlabeled (unobserved) documents for a given query $q$. The student model is then supervised by both ground-truth ranking from the training data set and teacher model's top-$K$ ranking on unlabeled documents.**

training data, we introduce extra information generated from a well-trained teacher model and make the student model as effective as the teacher.

## 3.1 Overview

Figure 3 shows the overview of ranking distillation. In the offline training phase (prior to any user query), similar to KD, first we train a large teacher model with a strong ranking performance on the training set. Then for each query, we use the well-trained teacher model to make predictions on unlabeled documents (green part in Figure 3) and use this extra information for learning the smaller student model. Since the teacher model is allowed to have many parameters, it captures more complex features for ranking and is much powerful, thus, its predictions on unlabeled documents could be used to provide extra information for the student model's training. The student model with fewer parameters is more efficient for online inference, and because of the extra information provided by the teacher model, the student model inherits the high ranking performance of the teacher model.

Specifically, the offline training for student model with ranking distillation consists of two steps. First, we train a larger teacher model $M_T$ by minimizing a ranking-based loss with the ground-truth ranking from the training data set, as showed in Eqn (1). With much more parameters in this model, it captures more patterns from data and thus has a strong performance. We compute the predicted relevance scores of the teacher model $M_T$ for unlabeled documents $\bar{O} = \{d : y_d = \emptyset\}$ and get a top-$K$ unlabeled document ranking $\boldsymbol{\pi}_{1..K} = (\pi_1, \ldots, \pi_K)$, where $\pi_r \in \mathcal{D}$ is the $r$-th document in this ranking. Then, we train a smaller ranking model $M_S$ to minimize a *ranking loss* from the ground-truth ranking in the training data set, as well as a *distillation loss* with the exemplary top-$K$ ranking on unlabeled document set $\boldsymbol{\pi}_{1..k}$ offered by its teacher $M_T$. The overall loss to be minimized is as follows:

$$\mathcal{L}(\theta_S) = (1-\alpha)\mathcal{L}^R(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \alpha\mathcal{L}^D(\boldsymbol{\pi}_{1..K}, \hat{\boldsymbol{y}}). \tag{4}$$

Here $\hat{\boldsymbol{y}}$ is the student model's predicted scores[1]. $\mathcal{L}^R(,)$ stands for the ranking-based objective as in Eqn (1). The distillation loss, denoted by $\mathcal{L}^D(,)$, uses teacher model's top-$K$ ranking on unlabeled documents to guide the student model learning. $\alpha$ is the hyper-parameter used for balancing these two losses.

For a given query, the top documents ranked by the well-trained teacher can be regarded to have a strong correlation to this query, although they are not labeled in the training set. For example, if a user watches many action movies, the teacher's top-ranked documents may contain some other action movies as well as some adventure movies because they are correlated. In this sense, the proposed RD lets the teacher model teach its student to find the correlations and capture their patterns, thus, makes the student more generalizable and perform well on unseen data in the future. We use the top-$K$ ranking from the teacher instead of the whole ranked list because the noisy ranking at lower positions tends to cause the student model to overfit its teacher and lose generalizability. Besides, only top positions are considered important for ranking problems. $K$ is a hyper-parameter that represents the trust level on the teacher during teaching, *i.e.*, how much we adopt from teacher.

The choice of the ranking loss $\mathcal{L}^R(,)$ follows from different models' preferences and we only focus on the second term $\mathcal{L}^D(,)$ in Eqn (4). A question is how much we should trust teacher's top-$K$ ranking, especially for a larger $K$. In the rest of the section, we consider this issue.

## 3.2 Incorporating Distillation Loss

We consider the point-wise ranking loss for binary relevance labels for performing distillation, we also tried the pair-wise loss and will discuss their pros and cons later. Similar to Eqn (2), we formalize distillation loss as:

$$\begin{aligned}\mathcal{L}^D(\boldsymbol{\pi}_{1..K}, \hat{\boldsymbol{y}}) &= -\sum_{r=1}^{K} w_r \cdot \log(P(rel=1|\hat{y}_{\pi_r})) \\ &= -\sum_{r=1}^{K} w_r \cdot \log(\sigma(\hat{y}_{\pi_r})),\end{aligned} \tag{5}$$

where $\sigma(\cdot)$ is the *sigmoid* function and $w_r$ is the weight to be discussed later. There are several differences compared to Eqn (2). First, in Eqn (5), we treat the top-$K$ ranked documents from the teacher model as positive instances and there is no negative instance. Recall that KD causes the student model to output a higher probability for the label "tiger" when the ground-truth label is "cat" because their features captured by the teacher model are correlated. Along this line, we want the student model to rank higher for teacher's top-$K$ ranked documents. As we mentioned above, for the given query, besides the ground-truth positive documents $\boldsymbol{y}^+$, teacher's top-$K$ ranked unlabeled documents are also strongly correlated to this query. These correlations are captured by the well-trained powerful teacher model in the latent space when using latent factor model or neural networks.

However, as $K$ increases, the relevance of the top-$K$ ranked unlabeld documents becomes weaker. Following the work of learning

[1]When using point-wise and pair-wise losses, we only need to compute the student's predictions for a subset of documents, instead of all documents, for a given query.

**Algorithm 1** Estimate Student's Ranking for $\pi_r$

**Require:** Student Model $M_S(q, d; \theta_S)$, unlabeled document set $\bar{O}$ for a given query $q$ and the hyper-parameter $\epsilon$
  $\hat{y}_{\pi_r} \leftarrow M_S(q, \pi_r; \theta_S)$
  Initialize $n = 0$
  **for** $t = 1, 2, ...\epsilon$ **do**
    Sample a document $d$ from $\bar{O}$ without replacement
    $\hat{y}_d \leftarrow M_S(q, d; \theta_S)$
    **if** $\hat{y}_d > \hat{y}_{\pi_r}$ **then**
      $n \leftarrow n + 1$
    **end if**
  **end for**
  $\hat{r}_{\pi_r} \leftarrow \lfloor \frac{n \times (|\bar{O}|-1)}{\epsilon} \rfloor + 1$
  **return** $\hat{r}_{\pi_r}$

from noise labels [26], we use a weighted sum over the loss on documents from $\boldsymbol{\pi}_{1..K}$ with weight $w_r$ on each position $r$ from 1 to $K$. There are two straightforward choices for $w_r$: $w_r = 1/r$ puts more emphasis on the top positions, whereas $w_r = 1/K$ weights each position equally. Such weightings are heuristic and pre-determined, may not be flexible enough to deal with general cases. Instead, we introduce two flexible weighting schemes, which were shown to be superior in our experimental studies.

*3.2.1 Weighting by Position Importance.* In this weighting scheme, we assume that the teacher predicted unlabeled documents at top positions are more correlated to the query and are more likely to the positive ground-truth documents, therefore, this weight $w^a$ should be inversely proportional to the rank:

$$w_r^a \propto r^{-1} \quad \text{and} \quad r \in [1, K], \tag{6}$$

where $r$ is the rank range from 1 to $K$. As pointed out above, this scheme pre-determines the weight. Rendle *et al* [29] proposed an empirical weight for sampling a single position from a top-$K$ ranking, following a geometric distribution:

$$w_r^a = \rho(1-\rho)^r \quad \text{and} \quad \rho \in (0, 1). \tag{7}$$

Following their work, we use a parametrized geometric distribution for weighting the position importance:

$$w_r^a \propto e^{-r/\lambda} \quad \text{and} \quad \lambda \in \mathbb{R}^+, \tag{8}$$

where $\lambda$ is the hyperparameter that controls the sharpness of the distribution, and is searched through the validation set. When $\lambda$ is small, this scheme puts more emphasis on top positions, and when $\lambda$ is large enough, the distribution becomes the uniform distribution. This parametrization is easy to implement and configurable to each kind of situation.

*3.2.2 Weighting by Ranking Discrepancy.* The weighting by position importance is static, meaning that the weight at the same position is fixed during training process. Our second scheme is dynamic that considers the discrepancy between the student-predicted rank and the teacher-predicted rank for a given unlabeled document, and uses it as another weight $w^b$. This weighting scheme allows the training to gradually concentrate on the documents in teacher's top-$K$ ranking that are not well-predicted by the student. The details are as follows.

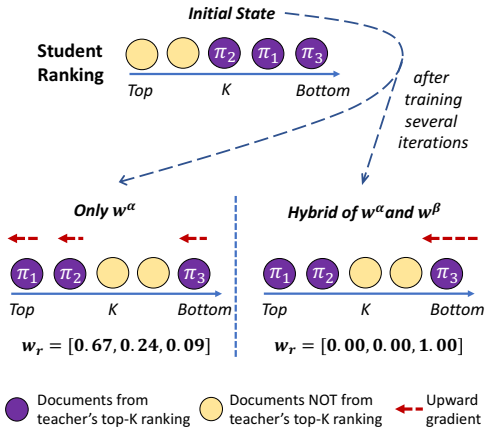



**Figure 4: An illustration of hybrid weighting scheme. We use $K = 3$ in this example.**

For the $r$-th document $\pi_r$ ($r \in [1, K]$) in teacher model's top-$K$ ranking, the teacher-predicted ranking (*i.e.,* $r$) is known for us. But we know only the student predicted relevant score $\hat{y}_{\pi_r}$ instead of its rank without computing relevance scores for all documents. To get the student predicted rank for this document, we apply Weston *et al* [36]'s sequential sampling, and do it in a parallel manner [16]. As described in Algorithm 1, for the $r$-th document $\pi_r$, if we want to know its rank in a list of $N$ documents without computing the scores for all documents, we can randomly sample $\epsilon \in [1, N-1]$ documents in this list and estimate the relative rank by $n/\epsilon$, where $n$ is the number of documents whose (student) scores are greater than $\hat{y}_{\pi_r}$. Then the estimated rank in the whole list is $\hat{r}_{\pi_r} = \lfloor \frac{n \times (N-1)}{\epsilon} \rfloor + 1$. When $\epsilon$ goes larger, the estimated rank is more close to the actual rank.

After getting the estimated student's rank $\hat{r}_{\pi_r}$ for the $r$-th document $\pi_r$ in teacher's top-$K$ ranking, the discrepancy between $r$ and $\hat{r}$ is computed by

$$w_r^b = tanh(\max(\mu \cdot (\hat{r}_{\pi_r} - r), 0)), \tag{9}$$

where $tanh(\cdot)$ is a rescaled *logistic* function $tanh(x) = 2\sigma(2x) - 1$ that rescale the output range to $[0, 1]$ when $x > 0$. The hyper-parameter $\mu \in \mathbb{R}^+$ is used to control the sharpness of the *tanh* function. Eqn (9) gives a dynamic weight: when the student predicted-rank of a document is close to its teacher, we think this document has been well-predicted and impose little loss on it (*i.e.,* $w_r^b \approx 0$); the rest concentrates on the documents (*i.e.,* $w_r^b \approx 1$) whose student predicted-rank is far from the teacher's rank. Note that the ranking discrepancy weight $w^b$ is computed for each document in $\pi_{1..K}$ during training. So in practice, we choose $\epsilon \ll |\hat{O}|$ for training efficiency. While extra computation used to compute relevance scores for sampled $\epsilon$ documents, we still boost the whole offline training process. Because the dynamic weight allows the training to focus on the erroneous parts in the distillation loss.

*3.2.3 Hybrid Weighting Scheme.* The hybrid weighting combines the weight $w^a$ by position importance, and the weight $w^b$ by ranking discrepancy: $w_r = (w_r^a \cdot w_r^b)/(\sum_{i=1}^K w_i^a \cdot w_i^b)$. Figure 4

**Table 1: Statistics of the data sets**

| Datasets | #users | #items | avg. actions per user | $(u, \mathcal{S}^{(u,t)})$ pairs | Sparsity |
|---|---|---|---|---|---|
| Gowalla | 13.1k | 14.0k | 40.74 | 367.6k | 99.71% |
| Foursquare | 10.1k | 23.4k | 30.16 | 198.9k | 99.87% |

illustrates the advantages of hybrid weighting over weighting only by position importance. Our experiments show that this hybrid weighting gives better results in general. In the actual implementation, since the estimated student ranking of $\hat{r}_{\pi_r}$ is not accurate during the first few iterations, we use only $w^a$ during the first $m$ iterations to warm up the model, and then use the hybrid weighting to make training focus on the erroneous parts in distillation loss. $m$ should be determined via the validation set. In our experiments, $m$ is usually set to more than half of the total training iterations.

### 3.3 Discussion

Under the paradigm of ranking distillation, for a certain query $q$, besides the labeled documents, we use a top-$K$ ranking for unlabeled documents generated by a well-trained teacher ranking model $M_T$ as extra information to guide the training of the student ranking model $M_S$ with less parameters. During the student model training, we use a weighted point-wise ranking loss as the distillation loss and propose two types of flexible weighting schemes, *i.e.*, $w^a$ and $w^b$ and propose an effective way to fusion them. For the hyper-parameters $(\alpha, \lambda, \mu, \epsilon, K, m)$, they are dataset-dependent and are determined for each data set through the validation set. Two key factors for the success of ranking distillation are: (1) larger models are capable to capture the complex interaction patterns between queries and documents, thus, their predicted unlabeled documents at top positions are also strongly correlated with the given query and (2) student models with less parameters can learn from the extra-provided helpful information (top-$K$ unlabeled documents in teacher's ranking) and boost their performances.

We also tried to use a pair-wise distillation loss when learning from teacher's top-$K$ ranking. Specifically, we use Eqn (3) for the distillation loss by taking the partial order in teacher's top-$K$ ranking as objective. However, the results were disappointing. We found that if we use pair-wise distillation loss to place much focus on the partial order within teacher's ranking, it will produce both upward and downward gradients, making the training unstable and sometimes even fail to converge. However, our weighted point-wise distillation loss that only contains upward gradients doesn't suffer from this issue.

## 4 EXPERIMENTAL STUDIES

We evaluate the performance of *ranking distillation* on two real-world data sets. The source code and processed data sets are publicly available online[2].

### 4.1 Experimental Setup

**Task Description**. We use recommendation as our task for evaluating the performance of ranking distillation. In this problem, we have a set of users $\mathcal{U} = \{u_1, u_2, \cdots, u_{|\mathcal{U}|}\}$ and a universe of items $\mathcal{I} = \{i_1, i_2, \cdots, i_{|\mathcal{I}|}\}$. For recommendation without context information, we can cache the recommendation list for each user [3]. However, for context-aware recommendation, we have to re-compute the recommendation list each time a user comes with a new context, so the online inference efficiency becomes important. The following *sequential recommendation* is one case of context-aware recommendation. Given a users $u$ with her/his history sequence (*i.e.,* past $L$ interacted items) at time $t$, $\mathcal{S}^{(u,t)} = (\mathcal{S}^u_{t-1}, .., \mathcal{S}^u_{t-L})$, where $\mathcal{S}^u_i \in \mathcal{I}$, the goal is to retrieve a list of items for this user that meets her/his future needs. In IR's terms, the query is the user profile $(u, \mathcal{S}^{(u,t)})$ at time $t$, and the document is the item. Note that whenever the user has a new behavior (*e.g.,* watch a video/listen to a music), we have to re-compute the recommendation list as her/his context changes. We also wish to point out that, in general, ranking distillation can be applied to other learning to rank tasks, not just to recommendation.

**Datasets**. We choose two real-world data sets in this work, as they contain numerous sequential signals and thus suitable for sequential recommendation [33]. Their statistics are described in Table 1. Gowalla[4] was constructed by [7] and Foursquare was obtained from [38]. These data sets contain sequences of implicit feedbacks through user-venue check-ins. During the offline training phase, for a user $u$, we extract every 5 successive items ($L = 5$) from her sequence as $\mathcal{S}^{(u,t)}$, and the immediately next item as the ground-truth. Following [33, 38], we hold the first 70% of actions in each user's sequence as the *training set* and use the next 10% of actions as the *validation set* to search the optimal hyperparameter settings for all models. The remaining 20% actions in each user's sequence are used as the *test set* for evaluating a model's performance.

**Evaluation Metrics**. As in [10, 27, 28, 37], three different evaluation metrics used are Precision@$n$ (Prec@$n$), nDCG@$n$, and Mean Average Precision (MAP). We set $n \in \{3, 5, 10\}$, as recommendations are top positions of rank lists are more important. To measure the online inference efficiency, we count the number of parameters in each model and report the wall time for making a recommendation list to every user based on her/his last 5 actions in the training data set. While training models efficiently is also important, training is done offline before the recommendation phase starts, and our focus is the online inference efficiency where the user is waiting for the responses from the system.

**Teacher/Student Models**. We apply the proposed ranking distillation to two sequential recommendation models that have been shown to have strong performances:

- **Fossil**. Factorized Sequential Prediction with Item Similarity ModeLs (Fossil) [13] models sequential patterns and user

[2] https://github.com/graytowne/rank_distill

[3] We suppose the recommendation model doesn't change immediately whenever new observed data come, which is common in real-world cases.

[4] https://snap.stanford.edu/data/loc-gowalla.html

**Table 2: Performance comparison. (1) The performance of the models with ranking distillation, Fossil-RD and Caser-RD, always has statistically significant improvements over the student-only models, Fossil-S and Caser-S. (2) The performance of the models with ranking distillation, Fossil-RD and Caser-RD, has no significant degradation from that of the teacher models, Fossil-T and Caser-T. We use the one-tail t-test with significance level at 0.05.**

**Gowalla**

| Model | Prec@3 | Prec@5 | Prec@10 | nDCG@3 | nDCG@5 | nDCG@10 | MAP |
|---|---|---|---|---|---|---|---|
| *Fossil-T* | 0.1299 | 0.1062 | 0.0791 | 0.1429 | 0.1270 | 0.1140 | 0.0866 |
| *Fossil-RD* | 0.1355 | 0.1096 | 0.0808 | 0.1490 | 0.1314 | 0.1172 | 0.0874 |
| *Fossil-S* | 0.1217 | 0.0995 | 0.0739 | 0.1335 | 0.1185 | 0.1060 | 0.0792 |
| *Caser-T* | 0.1408 | 0.1149 | 0.0856 | 0.1546 | 0.1376 | 0.1251 | 0.0958 |
| *Caser-RD* | 0.1458 | 0.1183 | 0.0878 | 0.1603 | 0.1423 | 0.1283 | 0.0969 |
| *Caser-S* | 0.1333 | 0.1094 | 0.0818 | 0.1456 | 0.1304 | 0.1188 | 0.0919 |
| *POP* | 0.0341 | 0.0362 | 0.0281 | 0.0517 | 0.0386 | 0.0344 | 0.0229 |
| *ItemCF* | 0.0686 | 0.0610 | 0.0503 | 0.0717 | 0.0675 | 0.0640 | 0.0622 |
| *BPR* | 0.1204 | 0.0983 | 0.0726 | 0.1301 | 0.1155 | 0.1037 | 0.0767 |

**Foursquare**

| Model | Prec@3 | Prec@5 | Prec@10 | nDCG@3 | nDCG@5 | nDCG@10 | MAP |
|---|---|---|---|---|---|---|---|
| *Fossil-T* | 0.0859 | 0.0630 | 0.0420 | 0.1182 | 0.1085 | 0.1011 | 0.0891 |
| *Fossil-RD* | 0.0877 | 0.0648 | 0.0430 | 0.1203 | 0.1102 | 0.1023 | 0.0901 |
| *Fossil-S* | 0.0766 | 0.0556 | 0.0355 | 0.1079 | 0.0985 | 0.0911 | 0.0780 |
| *Caser-T* | 0.0860 | 0.0650 | 0.0438 | 0.1182 | 0.1105 | 0.1041 | 0.0941 |
| *Caser-RD* | 0.0923 | 0.0671 | 0.0444 | 0.1261 | 0.1155 | 0.1076 | 0.0952 |
| *Caser-S* | 0.0830 | 0.0621 | 0.0413 | 0.1134 | 0.1051 | 0.0986 | 0.0874 |
| *POP* | 0.0702 | 0.0477 | 0.0304 | 0.0845 | 0.0760 | 0.0706 | 0.0636 |
| *ItemCF* | 0.0248 | 0.0221 | 0.0187 | 0.0282 | 0.0270 | 0.0260 | 0.0304 |
| *BPR* | 0.0744 | 0.0543 | 0.0348 | 0.0949 | 0.0871 | 0.0807 | 0.0719 |

preferences by fusing a similarity model with latent factor model. It uses a pair-wise ranking loss.

- **Caser**. ConvolutionAl Sequence Embedding Recommendation model (Caser) [33] incorporates the Convolutional Neural Network and latent factor model to learn sequential patterns as well as user preferences. It uses a point-wise ranking loss.

To apply ranking distillation, we adopt as many parameters as possible for the teacher model to achieve a good performance on each data set. These well-trained teacher models are denoted by **Fossil-T** and **Caser-T**. We then use these models to teach smaller student models denoted by **Fossil-RD** and **Caser-RD** by minimizing the ranking distillation loss in Eqn (4). The model sizes of the student models are gradually increased until the models reach a comparable performance to their teachers. **Fossil-S** and **Caser-S** denote the student models trained with only ranking loss, *i.e.*, without the help from the teacher. Note that the increasing in model sizes is achieved by using larger dimensions for embeddings, without any changes to the model structure.

## 4.2 Overall Results

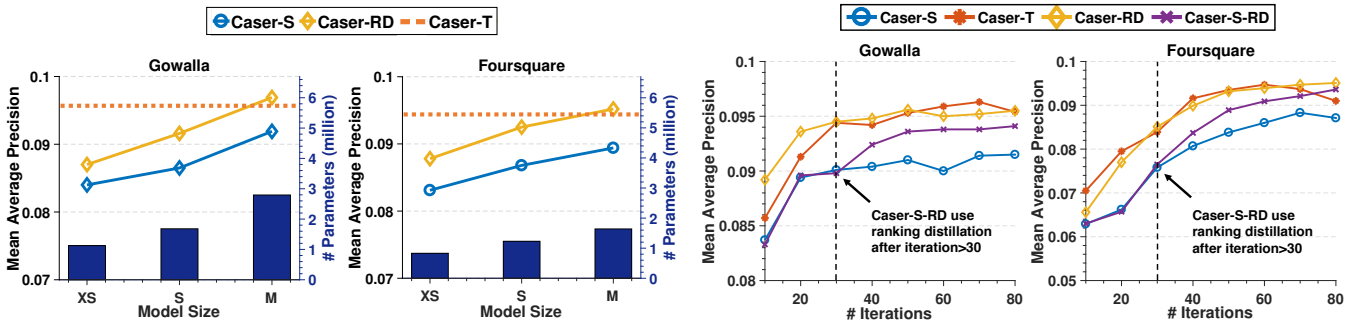
The results of each method are summarized in Table 2. We also included three non-sequential recommendation baselines: the popularity (in all users' sequences) based item recommendation (**POP**), the item based Collaborative Filtering[5] (**ItemCF**) [32], and the Bayesian personalized ranking (**BPR**) [30]. Clearly, the performance of these non-sequential baselines is worse than that of the sequential recommenders, *i.e.,* Fossil and Caser.

The teacher models, *i.e.*, Fossil-T and Caser-T, have a better performance than the student-only models, *i.e.*, Fossil-S and Caser-S, indicating that a larger model size provides more flexibility to fit the complex data with more predictive power. The effectiveness of ranking distillation is manifested by the significantly better performance of Fossil-RD and Caser-RD compared to Fossil-S and

[5]We use Jaccard similarity measure and set the number of nearest neighbor to 20.

**Table 3: Model compactness and online inference efficiency. Time (seconds) indicates the wall time used for generating a recommendation list for every user. Ratio is the student model's parameter size relative to the teacher model's parameter size.**

| Datasets | Model | Time (CPU) | Time (GPU) | #Params | Ratio |
|---|---|---|---|---|---|
| Gowalla | *Fossil-T* | 9.32s | 3.72s | 1.48M | 100% |
| | *Fossil-RD* | 4.99s | 2.11s | 0.64M | 43.2% |
| | *Caser-T* | 38.58s | 4.52s | 5.58M | 100% |
| | *Caser-RD* | 18.63s | 2.99s | 2.79M | 50.0% |
| Foursquare | *Fossil-T* | 6.35s | 2.47s | 1.01M | 100% |
| | *Fossil-RD* | 3.86s | 2.01s | 0.54M | 53.5% |
| | *Caser-T* | 23.89s | 2.95s | 4.06M | 100% |
| | *Caser-RD* | 11.65s | 1.96s | 1.64M | 40.4% |




(a) MAP vs. model size

(b) MAP vs. the number of iterations for model training

**Figure 5: Mean average precision vs. (a) model size and (b) the choice of distillation loss.**

Caser-S, and by the similar performance of Fossil-RD and Caser-RD compared to Fossil-T and Caser-T. In other words, thanks to the knowledge transfer of ranking distillation, we are able to learn a student model that has fewer parameters but similar performance as the teacher model. Surprisingly, student models with ranking distillation often have even better performance than their teachers. This finding is consistent with [20] and we will explain possible reasons in Section 4.3.

The online inference efficiency is measured by the model size ( number of model parameters) and is shown in Table 3. Note that Fossil-S and Caser-S have the same model size as Fossil-RD and Caser-RD. All inferences were implemented using PyTorch with CUDA from GTX1070 GPU and Intel i7-6700K CPU. Fossil-RD and Caser-RD nearly half down the model size compared to their teacher models, Fossil-T and Caser-T. This reduction in model size is translated into a similar reduction in online inference time. In many practical applications, the data set is much larger than the data sets considered here in terms of the numbers of users and items; for example, Youtube could have 30 million active users per day and 1.3 billion of items[6]. For such large data sets, online inference could be more time-consuming and the reduction in model size has more privileges. Also, for models that are much more complicated than Fossil and Caser, the reduction in model size could yield a larger reduction in online inference time than reported here.

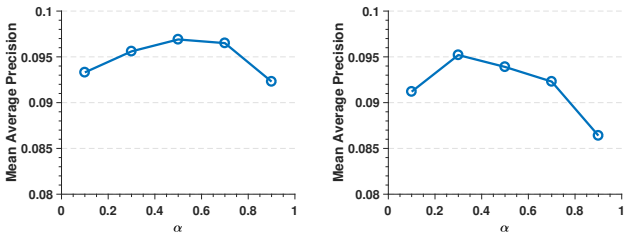
In conclusion, the findings in Table 2 and 3 together confirm that ranking distillation helps generate compact models with no or little compromise on effectiveness, and these advantages are independent of the choices of models.

## 4.3 Effects of Model Size and Distillation Loss

In this experiment, we study the impact of model size on the student model's performance (*i.e.,* MAP). We consider only Caser because the results for Fossil are similar. Figure 5a shows the results. Caser-S and Caser-RD perform better when the model size goes up, but there is always a gap. Caser-RD reaches a similar performance to its teacher with the medium model size, which is about 50% of the teacher model size.

Figure 5b shows the impact of ranking distillation on the student model's MAP iteration by iteration. We compare four models:

[6]https://fortunelords.com/youtube-statistics




(a) Gowalla

(b) Foursquare

**Figure 6: MAP vs. balancing parameter $\alpha$**

Case-S, Caser-T, Caser-RD, and Caser-S-RD. The last model minimizes ranking loss during the first 30 iterations and adds distillation loss after that. Caser-RD outperforms Caser-S all the time. Caser-S reaches its limit after 50 iterations on Gowalla and 70 iterations on Foursquare. Caser-S-RD performs similarly to Caser-S during the first 30 iterations, but catches up with the Caser-RD at the end, indicating the impressive effectiveness of ranking distillation. Caser-T performs well at first but tends to get overfitted after about 60 iterations due to its large model size and the sparse recommendation data sets. In contrast, Caser-RD and Caser-S-RD, which have smaller model sizes, are more robust to overfitting issue, though their training is partially supervised by the teacher. This finding reveals another advantage of ranking distillation.

Figure 6 shows the MAP for various balancing parameter $\alpha$ to explore models' performance when balancing ranking loss and distillation loss. For Gowalla data, the best performance is achieved when $\alpha$ is around 0.5. But for Foursquare data, the best performance is achieved when $\alpha$ is around 0.3, indicating too much concentrate on distillation loss leads to a bad performance. On both data sets, either discarding ranking loss or discarding distillation loss gives poor results.

## 4.4 Effects of Weighting Schemes

Table 4 shows the effects of the proposed weighting schemes in our ranking distillation. For the weight $w_r$ for $r$-th document in the teacher's top-$K$ ranking, we used the equal weight ($w_r = 1/K$) as the baseline and considered the weighting by position importance

**Table 4: Performance of Caser-RD with different choices of weighting scheme on two data sets.**

| Datasets | Weighting | P@10 | nDCG@10 | MAP |
|---|---|---|---|---|
| Gowalla | $w_r = 1/K$ | 0.0843 | 0.1198 | 0.0925 |
| | $w_r = w_r^a$ | 0.0850 | 0.1230 | 0.0945 |
| | $w_r = w_r^b$ | 0.0851 | 0.1227 | 0.0937 |
| | hybrid | **0.0878** | **0.1283** | **0.0969** |
| Foursquare | $w_r = 1/K$ | 0.0424 | 0.1046 | 0.0914 |
| | $w_r = w_r^a$ | 0.0423 | 0.1052 | 0.0929 |
| | $w_r = w_r^b$ | 0.0429 | 0.1035 | 0.0912 |
| | hybrid | **0.0444** | **0.1076** | **0.0952** |

($w_r = w_r^a$), the weighting by ranking discrepancy ($w_r = w_r^b$), and the hybrid weighting ($w_r \propto w_r^a \cdot w_r^b$). The equal weight performs the worst. The position importance weighting is much better, suggesting that within the teacher's top-$K$ ranking, documents at top positions are more related to the positive ground truth. The ranking discrepancy weighting only doesn't give impressive results, but when used with the position importance weighting, the hybrid weighting yields the best results on both data sets.

## 5 RELATED WORK

In this section, we compared our works with several related research areas.

**Knowledge Distillation** Knowledge distillation has been used in image recognition [3, 15, 31] and neural machine translation [20] as a way to generate compact models. As pointed out in Introduction, it is not straightforward to apply KD to ranking models and new issues must be addressed. In the context of ranking problems, the most relevant work is [6], which uses knowledge distillation for image retrieval. This method applies the sampling technique to rank a sample of the image from all data each time. In general, training on a sample works if the sample shares similar patterns with the rest of data through some content information, such as image contents in the case of [6]. But this technique is not applicable to training a recommender model when items and users are represented by IDs with no content information, as in the case of collaborative filtering. In this case, the recommender model training cannot be easily generalize to all users and items.

**Semi-Supervised Learning** Another related research area is semi-supervised learning [4, 43]. Unlike the teacher-student model learning paradigm in knowledge distillation and in our work, semi-supervised learning usually trains a single model and utilizes weak-labeled or unlabeled data as well as the labeled data to gain a better performance. Several works in information retrieval followed this direction, using weak-labeled or unlabeled data to construct test collections [2], to provide extra features [11] and labels [10] for ranking model training. The basic idea of ranking distillation and semi-supervised learning is similar as they both utilize unlabeled data while with different purpose.

**Transfer Learning for Recommender System** Transfer learning has been widely used in the field of recommender systems [5, 12]. These methods mainly focus on how to transfer knowledge (*e.g.*, user rating patterns) from a source domain (*e.g.*, movies) to a target domain (*e.g.,* musics) for improving the recommendation performance. If we consider the student as a target model and the teacher as a source model, our teacher-student learning can be seen as a special transfer learning. However, unlike transfer learning, our teacher-student learning does not require two domains because the teacher and student models are learned from the same domain. Having a compact student model to enhance online inference efficiency is another purpose of our teacher-student learning.

## 6 CONCLUSION

The proposed ranking distillation enables generating compact ranking models for better online inference efficiency without scarifying the ranking performance. The idea is training a teacher model with more parameters to teach a student model with fewer parameters to rank unlabeled documents. While the student model is compact, its training benefits from the extra supervision of the teacher, in addition to the usual ground truth from the training data, making the student model comparable with the teacher model in the ranking performance. This paper focused on several key issues of ranking distillation, *i.e.*, the problem formulation, the representation of teacher's supervision, and the balance between the trust on the training data and the trust on the teacher, and presented our solutions. The evaluation on real data sets supported our claims.

## ACKNOWLEDGEMENT

The work of the second author is partially supported by a Discovery Grant from Natural Sciences and Engineering Research Council of Canada.

## REFERENCES

[1] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormándi, George E. Dahl, and Geoffrey E. Hinton. 2018. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235* (2018).
[2] Nima Asadi, Donald Metzler, Tamer Elsayed, and Jimmy Lin. 2011. Pseudo test collections for learning web search ranking functions. In *International Conference on Research and development in Information Retrieval*. ACM, 1073–1082.
[3] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *Advances in neural information processing systems*. 2654–2662.
[4] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. 2009. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks* 20, 3 (2009), 542–542.
[5] Xu Chen, Zheng Qin, Yongfeng Zhang, and Tao Xu. 2016. Learning to rank features for recommendation over multiple categories. In *International ACM SIGIR conference on Research and Development in Information Retrieval*. 305–314.
[6] Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. 2017. DarkRank: Accelerating Deep Metric Learning via Cross Sample Similarities Transfer. *arXiv preprint arXiv:1707.01220* (2017).
[7] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *International Conference on Knowledge Discovery and Data Mining*. ACM, 1082–1090.
[8] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. 2015. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*. 192–204.
[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *ACM Conference on Recommender systems*. 191–198.
[10] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. *arXiv preprint arXiv:1704.08803* (2017).
[11] Fernando Diaz. 2016. Learning to Rank with Labeled Features. In *International Conference on the Theory of Information Retrieval*. ACM, 41–44.
[12] Ignacio Fernández-Tobías, Iván Cantador, Marius Kaminskas, and Francesco Ricci. 2012. Cross-domain recommender systems: A survey of the state of the art. In *Spanish Conference on Information Retrieval*. sn, 24.

[13] Ruining He and Julian McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In *International Conference on Data Mining*. IEEE.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *International Conference on World Wide Web*. ACM, 173–182.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[16] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *International Conference on World Wide Web*. 193–201.

[17] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *IEEE conference on Computer Vision and Pattern Recognition*. 1725–1732.

[18] Kenji Kawaguchi. 2016. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*. 586–594.

[19] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Conference on Empirical Methods on Natural Language Processing*. ACL, 1756–1751.

[20] Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947* (2016).

[21] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[23] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems. In *International Conference on Management of Data*. ACM, 835–850.

[24] David C Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related pins at pinterest: The evolution of a real-world recommender system. In *International Conference on World Wide Web*. 583–592.

[25] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*.

[26] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. 2013. Learning with noisy labels. In *Advances in neural information processing systems*. 1196–1204.

[27] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching As Image Recognition. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2793–2799.

[28] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. 2017. DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval. *arXiv preprint arXiv:1710.05649* (2017).

[29] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *International Conference on Web Search and Data Mining*. ACM, 273–282.

[30] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.

[31] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[32] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*. ACM, 285–295.

[33] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *ACM International Conference on Web Search and Data Mining*.

[34] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. Lemp: Fast retrieval of large entries in a matrix product. In *International Conference on Management of Data*. ACM, 107–122.

[35] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *International Conference on Knowledge Discovery and Data Mining*. ACM, 1235–1244.

[36] Jason Weston, Samy Bengio, and Nicolas Usunier. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning* 81, 1 (2010), 21–35.

[37] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *International ACM SIGIR conference on Research and Development in Information Retrieval*. 55–64.

[38] Quan Yuan, Gao Cong, and Aixin Sun. 2014. Graph-based point-of-interest recommendation with geographical and temporal influences. In *International Conference on Information and Knowledge Management*. ACM, 659–668.

[39] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *International Conference on Research and Development in Information Retrieval*. ACM, 325–334.

[40] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete Personalized Ranking for Fast Collaborative Filtering from Implicit Feedback.. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 1669–1675.

[41] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *International Conference on Research and Development in Information Retrieval*. ACM, 183–192.

[42] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *International Conference on Knowledge Discovery and Data Mining*. ACM, 498–506.

[43] Xiaojin Zhu. 2005. Semi-supervised learning literature survey. (2005).