FedFast: Going Beyond Average for Faster Training of Federated Recommender Systems

Khalil Muhammad*, Qinqin Wang*, Diarmuid O'Reilly-Morgan*, Elias Tragos*, Barry Smyth*, Neil Hurley*, James Geraci†, Aonghus Lawlor*
{first.last}@insight-centre.org;james.geraci@samsung.com
*Insight Centre for Data Analytics, University College Dublin, Dublin, Ireland
†Samsung Electronics Co., Ltd., Seoul, Republic of Korea

ABSTRACT

Federated learning (FL) is quickly becoming the de facto standard for the distributed training of deep recommendation models, using on-device user data and reducing server costs. In a typical FL process, a central server tasks end-users to train a shared recommendation model using their local data. The local models are trained over several rounds on the users' devices and the server combines them into a global model, which is sent to the devices for the purpose of providing recommendations. Standard FL approaches use randomly selected users for training at each round, and simply average their local models to compute the global model. The resulting federated recommendation models require significant client effort to train and many communication rounds before they converge to a satisfactory accuracy. Users are left with poor quality recommendations until the late stages of training. We present a novel technique, `FedFast`, to accelerate distributed learning which achieves good accuracy for all users very early in the training process. We achieve this by sampling from a diverse set of participating clients in each training round and applying an active aggregation method that propagates the updated model to the other clients. Consequently, with `FedFast` the users benefit from far lower communication costs and more accurate models that can be consumed *anytime* during the training process even at the very early stages. We demonstrate the efficacy of our approach across a variety of benchmark datasets and in comparison to state-of-the-art recommendation techniques.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

recommender systems, federated learning, active sampling, faster training, communication costs

ACM Reference Format:
Khalil Muhammad*, Qinqin Wang*, Diarmuid O'Reilly-Morgan*, Elias Tragos*, Barry Smyth*, Neil Hurley*, James Geraci†, Aonghus Lawlor*. 2020. `FedFast`: Going Beyond Average for Faster Training of Federated Recommender Systems. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403176

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403176

1 INTRODUCTION

Recommender systems (RS) help users discover the most useful information or service at the right time and in the most appropriate way. They do this by learning the tastes and preferences of users, thus creating a trade-off between the benefits of personalisation and the privacy costs that come with the reliance on personal data. Indeed, with recent changes in data privacy laws, increased regulation (e.g. GDPR), and a growing unease amongst the general public about how their personal data is collected and used, it has become increasingly important for RS to seek a better balance between personalisation and privacy [27].

Additionally, the requirements of training complex RS models for millions of users and items on a central server raises scalability issues and requires significant computing and storage resources, as well as high server maintenance costs for companies. Fortunately, Federated Learning [20] addresses these concerns by facilitating the decentralised training of RS. In such federated recommender systems, users collaboratively train the model without sharing their personal data with a centralised server or with other users. This means that training and inference can be executed locally on user devices using simple models. The opportunity of combining recent advances in neural recommender systems [6, 13] and FL [20] motivates the problem tackled in this paper.

Problem statement. Most existing applications of FL in recommender systems focus on improving both accuracy and privacy [2, 4]. While FL can train a global model without the users' data leaving their devices, the extent to which FL can be considered a privacy-preserving method is debatable since private information can still be inferred from the model parameters [21]. Another noteworthy issue with FL systems is that users tend to incur the majority of the costs involved in training the model. Specifically, there is a potential for users to experience performance degradation and an increase in communication payload during the training process. These issues can easily frustrate users and ultimately reduce the acceptance of the recommender system, especially if they have to wait for a long time to enjoy good quality, personalised recommendations. Hence, it is important for developers of FL algorithms to understand the trade-offs required to make faster and more accurate federated recommender systems.

Our motivation is to develop a more efficient method for the faster training of RS models in a federated setting so that users can enjoy accurate recommendations without expending unnecessary effort and enduring high communication costs.

Contribution. Thus, we propose FedFast, an *anytime* [11] framework that speeds up the convergence rate of recommendation models while guaranteeing that users benefit from more accurate recommendations even before the training process is complete. The agility of FedFast stems from two novel components: (i) ActvSAMP to better choose the users that participate in each round of training, and (ii) ActvAGG to combine locally trained models in a smart manner that speeds up the training process.

Scope. In this work we extend the familiar *Federated Averaging* algorithm FedAvg [20] to increase its convergence speed. Although we describe our proposed solution (FedFast) in the context of a relatively straightforward neural recommendation model (GMF), FedFast can easily be applied on more sophisticated models that rely on learned user and item embeddings. Accordingly, we evaluate FedFast on four real-world data sets to show its superiority over FedAvg and its competitiveness compared to centrally trained state of the art GMF [13] and BPR [24] models.

Overall, in this work we aim to answer these research questions: (RQ1) can FedFast consistently outperform FedAvg in recommendation quality across all *rounds* of training to quality as an *anytime* algorithm? (RQ2) Does FedFast converge faster than FedAvg to reach a similar or better recommendation accuracy on the same recommendation task? (RQ3) How sensitive is FedFast to its hyperparameters in terms of convergence speed and recommendation accuracy? (RQ4) how significant are contributions of ActvSAMP and ActvAGG to FedFast?, and finally (RQ5) does FedFast behave consistently across different datasets?

2 RELATED WORK

Matrix Factorisation (MF) [17] has provided a strong foundation for developing accurate collaborative filtering (CF) recommender systems for the last decade. Lately, there is a trend to improve MF by exploiting deep neural networks (DNNs) for modelling extra information, such as text, images, and sound [28]. Approaches for DNN in RS include [6, 12, 13]. Many of these approaches have significantly improved the performance of standard RS algorithms, however their main characteristic is that they are centralised, not scalable as the number of users and items increase and require users to share their personal data with the server, which then stores and processes them for training the DNN models.

To improve scalability and overcome the need for central collection of data, a new method for training machine learning models in a collaborative and distributed way has emerged, named *Federated Learning* (FL) [15, 20, 20]. It works by initialising a global model and starts an iterative process, where at the start of each round, the server selects a subset of existing clients to locally train and update the model with their on-device data. The clients send these updates to the server, which then aggregates the received updates to supplant the weights of the global model. This process continues until the model has been fully trained.

Despite FL being a promising technique, its application on RS has not received much attention. Recently, Chen et al. [4] proposed a federated meta-learning framework that shares user information at an *algorithm* level. This approach differs from the conventional FL algorithm [20] that adopts federated averaging to share information at a *model* level. Chen et al. [4] claim that their approach outperforms unified models built on [20] and standalone algorithms, in recommendation accuracy and is also more efficient in terms of model sizes. Even more recently, Ammad et al. [2] developed a federated collaborative filtering algorithm for personalised recommendations. In their work, the update to the the model is based on a stochastic gradient approach. By using an adaptive learning rate, they are able to achieve a similar recommendation performance as the standard FL method [20]. In general, it appears that most existing FL solutions in recommender systems aim to improve prediction accuracy. However, our incentive is to improve the training process, making it much faster so that the costs incurred by users (in terms of total battery consumption or performance degradation) is significantly decreased.

Therefore, we focus on extending [20] to make it more efficient when training a recommender system model. We propose a novel strategy for sampling only a subset of all available users to train a recommendation model without loss of accuracy. Recent works have demonstrated that effective sampling techniques can significantly improve the accuracy and training speed of classic [29] and deep [5] recommender systems. Since current FL implementations [3, 20] sample users randomly to participate in the training the model, we will describe ActvSAMP and ActvAGG as a more effective sampling strategy.

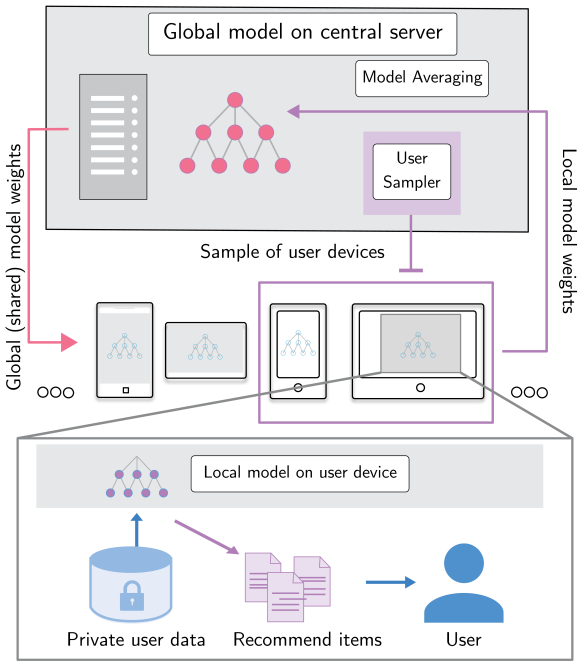

Figure 1: Architecture of FedFast showing the key components and the interactions that are common in a federated learning system

Like [2], our work is based on users' implicit feedback, whereas [4] is developed in the context of a rating prediction problem. We focus on improving both accuracy, and convergence speed during training. We expect that the data locality constraint would enhance users' privacy too. Interestingly, we are not aware of any existing published work that address this training efficiency issue. One first attempt has started in [10] which aims at sampling users at each round according to the probability that their data are useful in that round, achieving a small improvement in the training speed compared to the state of the art. However, this work requires all users to do a lot of extra work in order to calculate a valuation function for the usefulness of their data and to communicate this to the server *at each round*.

3 SYSTEM OVERVIEW

Our system, as illustrated in Figure 1, conforms to a client-server architecture that is familiar to federated learning systems. Each client is a user device that can train a lightweight recommendation model using locally stored private data. The server maintains a global copy of the recommendation model, and controls the distributed learning process using two key components: a *user sampler* and a *model averager*. Basically, the user sampler (`ActvSAMP`) dynamically chooses clients to partake in training the recommendation model, whereas the model averager aggregates the parameters learned by the clients using `ActvAGG`. We describe below in more depth the details of the recommendation model and the sampling algorithm.

3.1 The Recommendation Model

A core requirement of our proposed `FedFast` system is that the recommendation model is implemented as a neural network that predicts user preferences from the representations (user and item embeddings) it learns. It is also desirable for the recommendation model to be lightweight since its parameters will be frequently sent back-and-forth between the client and the server. Although a complex model (e.g. `NCF` [13]) is likely to provide more accurate predictions, it will have more parameters to learn, thus imposing communication and computation costs on the clients.

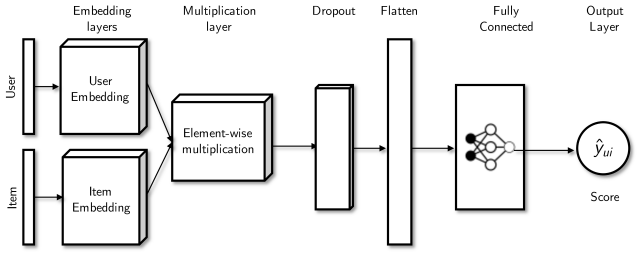

Figure 2: Architecture of General Matrix Factorisation, `GMF`

For this reason, we will evaluate our system using the General Matrix Factorization (`GMF`) model introduced in [13]. The `GMF` is a neural network based on matrix factorisation that predicts the user-item interaction $\hat{y}_{ui}$, see Figure 2.

The input layer consists of the user $\mathbf{v}_u^U$ and item $\mathbf{v}_i^I$ vectors that describe the user u and the item i, respectively. Then, a fully connected *embedding layer* projects the $\mathbf{v}_u^U$ and item $\mathbf{v}_i^I$ vectors (from the input layer) to their latent vector representations. Here, a *user embedding* is analogous to the user's latent vector $\mathbf{p}_u$, whereas an *item embedding* corresponds to the item's latent vector $\mathbf{q}_i$. The multiplication layer applies the mapping function $\phi_1(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u \odot \mathbf{q}_i$ to combine user and item preferences ($\odot$ denotes the element-wise product operator). The resulting product vector is fed into at least one fully-connected layer with a linear activation function to predict user preferences. `GMF` is trained by minimising an appropriate loss function between the predicted rating $\hat{y}_{ui}$ and its real value y_{ui} — we use *mean squared error* for explicit recommendation and *binary cross-entropy* for implicit recommendation.

Justifying lightweight recommendation models: The clients in federated learning can be *thin* (e.g. smartphones, IoT devices) or *fat* (laptops, servers). We prefer to consider a more challenging task the where clients are thin and have limited bandwidth and processors. Accordingly, we make a case for mobile-first recommendation models (similar to MobileNets [14]) designed to effectively maximise accuracy while considering the restricted resources for an on-device or embedded application. Such models are ideally small, low-latency, low-power models, parameterised to meet the resource liabilities for thin devices. Our proposed solution is tailored to deep models that learn user and item embeddings, so there is a resource-accuracy trade-off due the proportionality between the the model size and the embedding size. For example, in our experiments, the `GMF` model for ML100K[1] has 21,009 parameters and weighs 84.5 kB when the embedding size is 8. But when the embedding size is 9, the model size grows by 12.4% and the number of learnable parameters becomes 23,635. This minor increment of the embedding size only improves the *Hit Ratio* by 0.6% despite the model size growing by 12.4%.

3.2 Federating the Recommendation Model

We extend the `FedAvg` algorithm in [20] by introducing a novel sampling and aggregation strategy to reduce the communication rounds required to train the recommendation model.

Parameter Notation: Let K be the set of all clients. We write the full set of parameters of the recommendation model as w. The number of parameters depends on the model and includes user and item embeddings. For an index j, we write $w[j]$, to represent the j^{th} component of the parameter set, and generalise this notation to write $w[J]$ for the parameters indexed by any subset J of the index set. In particular, let U be the set of indices corresponding to all user embeddings, and for $k \in K$, we will write U_k for the indices corresponding to the user embeddings associated with client k, so that $U = \cup_{k \in K} U_k$. Similarly, write I to represent indices corresponding to the item embeddings and N to represent all other non-embedding indices, so that the full set of indices is $U \cup I \cup N$. We also refer to the item embedding indices that are updated by client k during a particular round of the algorithm as $I_k \subseteq I$.

The `FedFast` *algorithm:* We formalise our proposal as `FedFast`, presented as Algorithm 1. We train the selected recommendation model (GMF), with parameters w, over a number of rounds t, such that, at each round, a different subset $S_t \subset K$ of m clients is selected

[1] https://grouplens.org/datasets/movielens/100k/

(using a sampling technique we call ActvSAMP) to concurrently train a copy of the model using their local data. At Line 4, the parameter $\alpha \leq 1$, is the proportion of clients sampled on each round, using a partition $\mathcal{G}$ of the clients into $p \geq 1$ clusters. During local training (referred to as *ClientUpdate*), each client makes at least one *epoch* (or iteration) over their training data to independently improve the model, and returns its updated parameters, w_{t+1}^k and the number of training samples used, n_k. Then, at the end of each round, the server aggregates (using a technique we call ActvAGG) all the locally trained models w_{t+1}^k to update the global recommendation model. Note that during the aggregation, the users are re-partitioned and the new client partition is used on the subsequent round.

Algorithm 1: The FedFast algorithm

```
1  initialise w_0, P_0, p
2  initialse G_0 ← ClusterUsers(K, p, P_0)
3  foreach round t = 0, 1, ... do
4      m ← max(⌈αK⌉, 1)
5      S_t ← ActvSAMP(K, m, G_t)
6      foreach client k ∈ S_t in parallel do
7          w_{t+1}^k, n_k ← ClientUpdate(k, w_t)
8      end
9      γ ← exp(−t) % discount factor
10     w_{t+1}, G_{t+1} ← ActvAGG(S_t, w_{t+1}^k, w_t, n_k, p, γ)
11 end
```

We describe these extensions to the FedAvg algorithm in the following sections.

3.3 Active Sampling for Client Selection

Clients in a typical RS tend to be heterogeneous with users having diverse preferences about different items. Even so, it is not uncommon to find groups of clients that share similar characteristics. This knowledge guides most recommendation algorithms, yet it remains unexploited in FL for recommender systems. For instance, the FedAvg algorithm [20] naively samples clients at random.

The motivation for ActvSAMP is that, if users can be grouped based on their profile similarities, then each group can benefit from the training experience of their peers that participate in training the RS model. This within-group exchange of training experiences (that occurs on the server) should reduce client workloads, reduce communication rounds to the server, improve recommendation quality, and increase the convergence speed of the model. Note that, unlike existing sampling techniques [1, 19, 26], ActvSAMP occurs on the server and relies on client metadata that guarantees their anonymity; hence its suitability for federated learning scenarios.

The ActvSAMP algorithm, as formalised in Algorithm 2, depends on a partition $\mathcal{G}$ of the clients into clusters, obtained from the *ClusterUser* function. In our implementation, we use the k-means algorithm for clustering. Initially (cf. Line 2 of Algorithm 1), client representations, P_0, are formed using a set of summary statistics (e.g. mean and entropy) of their user profiles. Then ActvSAMP, randomly samples one client per cluster in $\mathcal{G}$ in a round-robin manner until there are m clients in the sampling set S_t. Initialising the FL approach in this way allows us to pick a fair sample that reflects the different user communities in the domain. In a practical setting, it might be more useful to cluster the clients in K using other privacy-preserving features (e.g. region, device type) that would result in clusters that reflect the ideal user groups in K. Notwithstanding, working without such additional features does not discount the effectiveness of our solution. We will evaluate the use of other features in future work.

Algorithm 2: ActvSAMP

```
Data  : K: set of all clients
        m: number of clients to sample from K
        G: cluster of clients.
Result: S, set of m clients from K
1 p ← |G|    % number of clusters in G
2 S ←randomly sample m/p distinct clients from each cluster
    in G;
3 return S
```

In subsequent rounds, ActvAGG returns a new partition of the clients, $\mathcal{G}$, where the clustering is carried out over the set of user embeddings, $w[U]$ (cf. Line 18 of Algorithm 3), ensuring that clusters consist of clients whose user embeddings are similar.

3.4 Combining trained models using ActvAGG

In conventional federated learning algorithms, e.g. FedAvg, the global model w_{t+1} is updated with the weighted sum of all received local models (see Equation 1), where n_k is the number of local training examples for client k.

$$w_{t+1} = \sum_{k \in S_t} \frac{n_k}{n_\sigma} w_{t+1}^k, \qquad n_\sigma = \sum_{k \in S_t} n_k \tag{1}$$

There is an opportunity to make this aggregation process more efficient. Parameter updates learned by any user A can be directly applied to a similar user B, thus accelerating the learning process for user B and consequently improving the overall efficiency of the federated system. Thus, by updating $\mathcal{G}$ at the end of each round, we maintain accurate clusters of clients whose user-embeddings are highly similar and propagate client updates to all other clients of the cluster that contains them. This is the essence of our aggregation strategy, ActvAGG.

ActvAGG combines local models to update the shared global model in a manner that causes the federated model to converge faster to a better recommendation quality. We refer to those clients that partake in model training on a particular round as *delegates*; and those that do not participate in the training process as *subordinate users* or simply *subordinates*. ActvAGG, as described in Algorithm 3, works by sharing the training progress of delegates with their subordinates. This sharing is particularly effective in accelerating learning during early rounds and becomes less important as the model converges. Hence, we control its impact using a discounting factor. The algorithm proceeds in the following steps:

Update of non-embedding components: Those components of the parameters w that do not correspond to either user or item embeddings are updated in the same manner as the FedAvg algorithm (cf. Line 3 of Algorithm 3).

Algorithm 3: ActvAGG algorithm

Data : S: sampled clients
$\forall k \in S, w^k$: model weights for clients
w_0: aggregated weights from previous round
n_k: number of training instances on client k
p: number of client clusters
γ: averaging discount (for subordinate updates)

Result: w: aggregated weights from w^k, $\mathcal{G}$: client cluster

1 $w \leftarrow w_0$
2 *% updating non-embedding components*
3 $w[N] \leftarrow \sum_{k \in S} \frac{n_k}{n_\sigma} w^k[N], \qquad n_\sigma = \sum_{k \in S} n_k$
4 *% updating item embeddings*
5 initialise $\forall k\ q^k[I] \leftarrow 0$ *% matrix of item embedding weights*
6 **foreach** $k \in S$ **do**
7 $\quad I_k \leftarrow \{\ell \in I \mid |w^k[\ell] - w_0[\ell]| > 0\}$
8 $\quad q^k[I_k] \leftarrow |w^k[I_k] - w_0[I_k]|$
9 **end**
10 **foreach** $i \in \bigcup_{k \in S} I_k$ **do**
11 $\quad w[i] \leftarrow \frac{1}{\sum_{k \in S} q^k[i]} \sum_{k \in S} q^k[i] w^k[i]$
12 **end**
13 *% updating delegate embeddings*
14 **foreach** $k \in S$ **do**
15 $\quad w[U_k] \leftarrow w^k[U_k]$
16 **end**
17 *% updating subordinate user embeddings*
18 $\mathcal{G} \leftarrow ClusterUsers(K, p, w[U])$
19 initialise $\delta[U] \leftarrow 0$
20 **foreach** $k \in S$ **do**
21 $\quad c_k \leftarrow getCluster(\mathcal{G}, k)$
22 $\quad$ **foreach** $s \in c_k \setminus S$ **do**
23 $\quad\quad \delta[U_s] \leftarrow \delta[U_s] + (w^k[U_k] - w_0[U_k])$
24 $\quad$ **end**
25 **end**
26 **foreach** $c \in \mathcal{G}$ **do**
27 $\quad$ **foreach** $s \in c \setminus S$ **do**
28 $\quad\quad w[U_s] \leftarrow w_0[U_s] + \gamma \frac{1}{|c \cap S|} \delta[U_s]$
29 $\quad$ **end**
30 **end**
31 **return** $w, \mathcal{G}$

Update of item embeddings: In Lines 5 to 12, the item embeddings are updated. Each item embedding component, i, of the aggregated parameters, $w[i]$, is formed as a weighted average of the parameter values $w^k[i]$ learned by each delegate, where each delegate's contribution is proportional to the difference between $w^k[i]$ and the aggregated value from the previous round, $w_0[i]$ (cf. Line 11). Note that delegate k only contributes to those components, I_k, that it has updated during this round. In general, the item-sets I_k overlap and the item embedding components, i, receive contributions from multiple clients.

Update of delegate embeddings: Since each user is assigned to a unique client, the user embedding parameters are simply updated as the values obtained from the delegate responsible for that user (cf. Line 15).

Re-partitioning: A new client partition, $\mathcal{G}$, is formed, using the newly obtained user embeddings $w[U]$ (cf. Line 18).

Update of subordinate embeddings: Each subordinate is updated using the user embeddings of all delegates that share its cluster. Specifically, for cluster $c \in \mathcal{G}$, the average, δ_c, of the differences between the user embeddings of the delegates assigned to that cluster (i.e. $w^k[U_k]$, for $k \in c$) and and the aggregated embedding from the previous round, $w_0[U_k]$, is obtained (Lines 20 to 25). Then, these average differences are multiplied by a discount factor, that decays as the learning progresses and are then added to the user embeddings of the subordinates (cf. Line 28).

3.5 Algorithm complexity

There is a concern that k-means is unsuitable for large-scale data due to its time complexity, which is $O(n^{dk+1})$, where n is the number of users to be clustered, k is the number of clusters, and d is the size of the dimensions or embedding size. We are also aware of the other limitations of k-means, such as: (i) it might not find the optimal set of clusters, (ii) it works best when the clusters are spherical, separable and of similar sizes, and (iii) it requires k to be provided beforehand. That said, the reader should note that our proposed system does not have a fixed dependency on k-means so we advocate for more scalable, optimal and efficient clustering algorithms, e.g. Lloyd's algorithm (its runtime is $O(nkdi)$, where i is the number of iterations needed until convergence) or [23].

Even if a more efficient clustering algorithm is employed, the algorithm complexity is bound to some cost incurred by the server in grouping users. As we will show in our experiments, this server-based cost is a trade-off for fewer communication rounds and less training effort from the clients.

4 EVALUATION

In this section, we evaluate FedFast with the aim of answering the research questions described in the introduction section.

Datasets. We experimented with four publicly accessible datasets: MovieLens 1M[2], MovieLens 100K[3], TripAdvisor [8], and Yelp [4]. We have summarised the key characteristics of the datasets in Table 1. All the datasets have been widely used in recommender systems literature to evaluate collaborative filtering algorithms.

Dataset	Interaction#	Item#	User#	Density
ML1M	1,000,209	3,706	6,040	4.5%
ML100K	100,000	1,628	943	6.3%
TripAdvisor	25,340	1,861	3,985	0.34%
Yelp	79,087	9,323	7,975	0.11%

Table 1: Statistics of the evaluation datasets

Note that the TripAdvisor dataset of hotel ratings was obtained from [8]. We pre-processed all evaluation datasets by only retaining users that have rated at least 5 items and, further, we transformed

[2] http://grouplens.org/datasets/movielens/1m/
[3] http://grouplens.org/datasets/movielens/100K/
[4] https://www.yelp.com/dataset/challenge

the explicit ratings into implicit signals where each entry is marked as 0 or 1 indicating whether the user has rated the item.

Evaluation Protocols. For performance evaluation, we use *leave-one-out* protocol, which is widely used [7]. For each user, we held-out their latest interaction for the test set and used the remaining data for training. Since it is too time-consuming to rank all items for every user during evaluation, we followed the common strategy [9, 16] that randomly samples 50 items that are not interacted with by the user, ranking the test item among the 50 items. The performance of a ranked list is judged by *Hit Ratio* (HR) [7] and *Normalized Discounted Cumulative Gain* (NDCG)[5]. We truncated the ranked list at 10 for both metrics. As such, the HR intuitively measures whether the test item is present on the *top*-10 list, and the NDCG accounts for the position of the hit by assigning higher scores to hits at top ranks. We calculate both metrics for each test user and report the average score. Unless stated otherwise, we use the same hyper-parameters of FedAvg in FedFast, except for p, the number of client clusters constructed from user embeddings.

Baselines. We compare our proposed FedFast solution with the popular FedAvg method using GMF as the recommendation model. Recall that FedFast is designed to improve FedAvg in its sampling (ActvSAMP) and aggregation (ActvAGG) components.

Our evaluation target is to compare convergence speed and recommendation quality in a federated context, so we leave out comparison with other classical centralised algorithms because they are not easily amenable to work with federated learning. And since our solution, FedFast, is scoped to embedding-based models, we also leave out autoencoder-based recommendation models such as [25]. Since the operations (e.g. model averaging) that combine distributed trained models can affect their recommendation performance, we highlight the centrally-trained BPR [24] and GMF as upper-bounds for the recommendation accuracy of FedFast and FedAvg.

4.1 Performance Comparison (RQ1, RQ5)

The aim of this experiment is twofold: (i) to compare the performance of our proposed FedFast with the standard FL method FedAvg, and (ii) to compare FedFast with a centrally trained (GMF) version of the model it federates. For completeness, we also compare the aforementioned algorithms with a centrally-trained BPR algorithm. These comparisons should allow the reader to understand the trade-offs in training distributed recommendation models, one of which is compromising recommendation accuracy for privacy due to data locality.

Note that we do not aim to set a new state-of-the-art performance record, but rather to show how FedFast outperforms FedAvg as a FL method. Furthermore, because of our self-imposed constraint to use lightweight models, we will deliberately force all algorithms to have the same embedding (or factor) size of 10. We also run FedAvg and FedFast with the same hyper-parameters, but our choice for FedFast's p hyper-parameter is the default ($p = 20$). Table 2 shows the performance of FedFast in comparison to the other baselines measured by HR@10 and NDCG@10, respectively.

[5]NDCG is a commonly used ranking-based metric that emphasises the importance of the top ranks by logarithmically discounting ranks

		Federated		Centralised	
		FedFast	**FedAvg**	**GMF**	**BPR**
ML100K	HR	**0.89**	0.79	0.91	***0.92***
	NDCG	**0.62**	0.51	**0.42**	0.30
ML1M	HR	**0.71**	0.60	0.61	***0.86***
	NDCG	**0.44**	0.35	0.34	0.29
TripAdvisor	HR	**0.50**	0.42	0.45	***0.51***
	NDCG	**0.26**	0.22	0.24	0.20
Yelp	HR	**0.62**	0.23	0.71	***0.81***
	NDCG	**0.36**	0.11	**0.45**	0.28

Table 2: Recommendation performance for FedFast, FedAvg and the centrally-trained baselines GMF and BPR. Convergence plots for these results are presented in Figure 3.

Looking at Table 2, it is immediately obvious that FedFast consistently outperforms the FedAvg federated baseline. In the ML100K, this improvement is marginal possibly due to the dataset's high density. We attribute FedFast's superiority over FedAvg to be caused by our proposed ActvSAMP and ActvAGG.

Recall that, in all experiments throughout this paper, both FedAvg and FedFast use GMF as their underlying recommendation model. The expectation is that the recommendation quality from a centrally trained GMF model would be an upper-bound for FedAvg and FedFast. Interestingly, there were cases where both FedAvg and FedAvg performed competitively with GMF. No surprise, FedFast also performed better than FedAvg in this comparison; and in some cases FedFast is better than GMF.

The BPR algorithm is arguably a state-of-the-art in ranking-based implicit recommendation tasks. Admittedly, BPR could be tuned better to improve its results, although what we report is within an acceptable range of its performance. Our FedFast algorithm outperforms BPR on NDCG in the ML1M and TripAdvisor datasets, but more often than not, BPR tends to have a better recommendation quality than other algorithms. Generally, these results tell us that training recommendation models in a distributed setup might afford us some benefits for data locality and scalability, but that comes at a cost of degraded recommendation quality. We further conducted one-sample paired t-tests, verifying that all improvements are statistically significant for $p < 0.01$

4.2 Convergence Analysis (RQ2, RQ5)

To validate our claim of improved training efficiency, we compare the convergence of FedFast with FedAvg that does not use ActvSAMP and ActvAGG. For this comparison, we set up the experiment as described in Section 4.1, and the results are shown in Figure 3.

Both FedAvg and FedFast models start from the same HR@10 and NDCG@10 and FedAvg slowly converges towards it best values as presented in subsection 4.1. But FedFast converges much more quickly than FedAvg and settles to within 5% of its best values in all evaluation datasets. In ML100K, FedFast reached FedAvg's highest HR@10 at round 30. This means that clients of FedAvg will have to endure 94% more communication *rounds* and expend 94% more local training effort to enjoy the same performance as the clients of FedAvg. Naturally, this significant difference in convergence

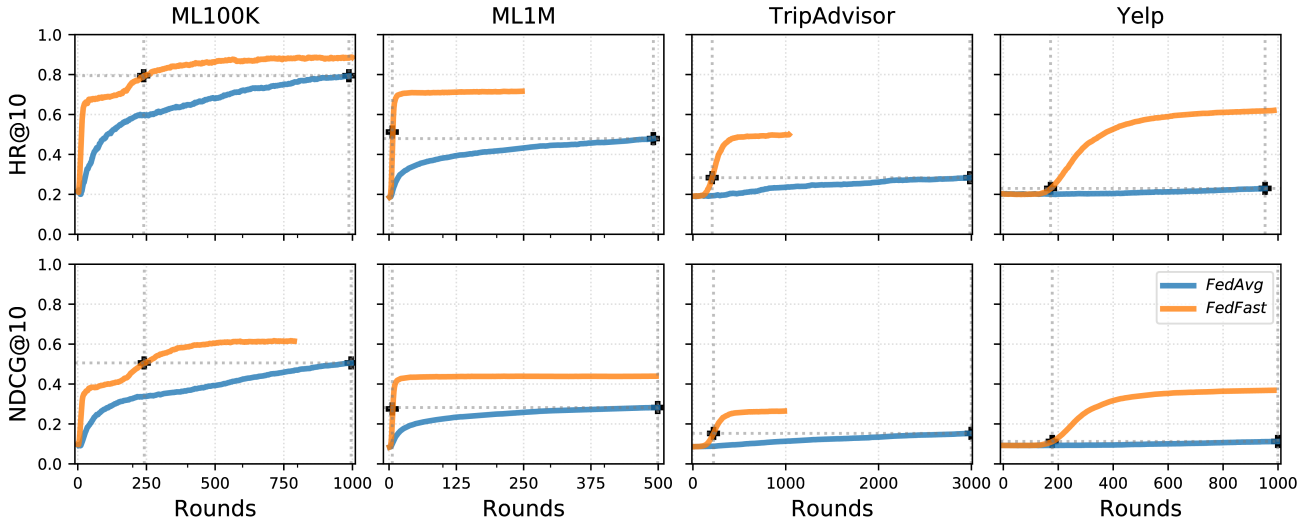

Figure 3: Convergence speed between FedFast and FedAvg on the evaluation datasets showing NDCG@10 and HR@10

speed can be attributed to the impact of ActvSAMP and ActvAGG in FedFast. Interestingly, across all rounds and in all evaluation datasets, FedFast consistently produces a better model than the FedAvg baseline. Empirically, this characteristic allows FedFast to be classified as an *anytime algorithm* [11]. As a matter of future work, we will further enhance FedFast to reduce the total number of users required to train the model across all training rounds.

It is also important to disambiguate between the time it takes to train a federated model and the number of rounds it takes to reach an optimal recommendation performance. These quantities are not necessarily correlated. For instance, in a worst-case scenario, the clustering process in FedFast could last longer than it takes for FedAvg to complete its rounds. Of course, this example is purely hypothetical to draw an example for the reader. We have left this analysis between training time and communication rounds for future work.

4.3 Sensitivity of FedFast to number of clusters (RQ3)

This experiment demonstrates the sensitivity of FedFast to the number of clusters (p) hyper-parameter. We ran FedFast using the protocol described in subsection 4.1 but with different values for the number of clusters, i.e. with $p \in \{5, 10, 20, 40, 60, 100\}$. The result of this experiment is presented in Figure 4.

It is immediately obvious that FedFast achieves similar recommendation quality (measured by HR@10 and NDCG@10) regardless of the number of clusters it takes as a hyper-parameter. The noticeable difference is how fast (measured by *Rounds*) the different models get to their best performance. For HR@10, it seems like the number of rounds is proportional to the size of p. But when $p = 100$, it takes much fewer rounds than when $p = 5$. Whereas, with NDCG@10, there is no discernible pattern: extreme values for p cause FedFast to converge slower than otherwise. Our offline

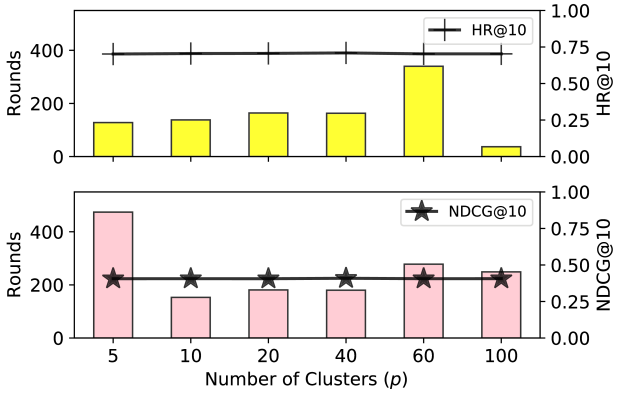

Figure 4: Number of round required for FedFast to achieve best HR@10 and NDCG@10 using different number of clusters p on the ML100K dataset.

analysis reveals that the ML100K can at best be partitioned into about 30 clusters using user metadata not fed to the FedFast model.

However, since FedFast clusters users (i.e. clients) based on their latent representations, it is unlikely the expected number of clusters (i.e. 30) is preserved in the latent space. Notwithstanding, the results in Figure 4 empirically suggest that the best value for p is closer to the number of clusters evaluated offline. This observation also holds true for the other datasets not presented here due to space constraints. Overall, this experiment shows that FedFast is likely to converge faster than FedAvg regardless of the value of p. A more thorough evaluation of this observation is deferred for future work.

4.4 Analysis of the individual impact of ActvSAMP and ActvAGG (RQ4)

This section analyses the individual contributions of the sampling (ActvSAMP) and aggregation (ActvAGG) components of FedFast. Focusing on the ML100K dataset due to space constraints, we use the results from the experiments described in Section 4.1 to show a more nuanced view into the enhancements introduced by FedFast over FedAvg in Figure 5.

FedFast is represented as the red line starting from a HR@10 of 0.21 and ending at 0.89, whereas FedAvg is the blue dashed line starting also from the same position as FedFast but ending as a 0.79 after 1000 rounds — that is 11% less. When FedAvg is combined with ActvSAMP, there is hardly any impact on FedAvg. This is because although ActvSAMP carefully selects good candidates to train the model, FedAvg's default aggregation strategy discards the most potential gains in performance introduced by ActvSAMP.

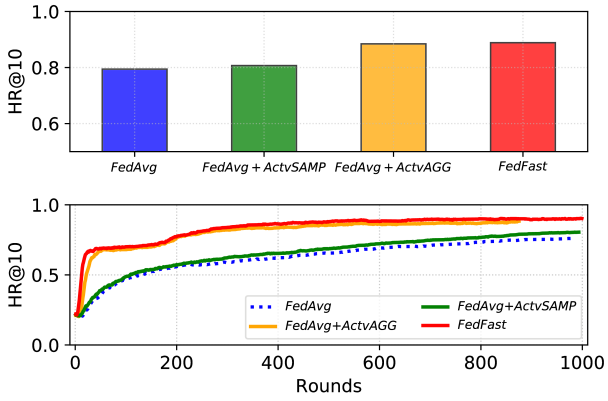

Figure 5: Contributions of the individual components of FedFast as an improvement over FedAvg in the ML100K dataset. FedAvg is almost identical to FedAvg + ActvSAMP

More apparent is the ActvAGG's contribution to FedAvg (depicted with the orange line). Impressively, it appears ActvAGG has more impact on increasing the recommendation quality and also on speeding up performance. Even so, FedFast still outperforms the combination of FedAvg and ActvAGG, being even faster to reach a good accuracy level.

In summary, what these results tell us is that ActvSAMP and ActvAGG are complementary to one another, hence the superiority of FedFast over FedAvg.

5 DISCUSSION

So far, we have described a strategy for making recommendations using federated learning recommendation models. We presented evaluations that prove the performance improvement of our approach, FedFast, in making accurate and efficient recommendations in a federated setting: FedFast can offer statistically significant improvements over the baselines we tested in evaluations.

Specifically, FedFast employs a novel sampling and aggregation technique to drastically reduce the number of communication rounds and clients effort involved in the training of federated recommender models. The empirical results prove this in four datasets, showing that the proposed algorithm is dataset agnostic. Although not presented in this work, the FedFast technique can also be used with other embedding-based recommendation models, such as NCF [13]. It also has to be noted that the faster training achievement does not compromise accuracy, since we have shown that FedFast outperforms the baseline FedAvg and compares with BPR in some cases. While these results are consistent across a variety of common evaluation datasets, we recognise that it would be appropriate to demonstrate these benefits in the context of a real-life system; this requires a live-study, but remains a key objective for future work.

It is noteworthy that the GMF recommendation model used throughout this paper is relatively simple. This choice is deliberate because its simplicity means that it will require less computational effort to be trained on a user's device. Consequently, FedFast and FedAvg are limited by the recommendation performance obtainable from GMF. The methods presented here, nevertheless, can be extended to capture more complexities [6] and to leverage additional data properties, e.g. [9, 18, 22]. We also expect FedFast to work seamlessly with other recommendation models that are based on deep neural networks that profile recommendation entities as embeddings.

The proposed FedFast algorithm, similar to all FL algorithms, requires users to do some extra work and train locally a deep recommendation model using their own data. The users' effort per round is not increased in FedFast compared to the standard FL models, since users only perform the task of training the model. In the end, since the training process finishes much faster (i.e. in fewer communication rounds) than baseline FL models, users are expected to save a lot of communication and processing resources using our model. In addition, the ActvSAMP technique is also selecting users from each cluster on a round-robin fashion in order to evenly distribute the load of the local training to users. The server in the FedFast algorithm is required to do some small extra work for clustering the users, which is alleviated by the amount of resources saved with the much faster training process.

In the FL setting, users might have sporadic online presence, which means that some might be unable to participate in training the model for long periods of time. If these sporadic users are ones that contribute a lot to the model training, this could have a significant effect on the model performance in the baseline scenario. However, in FedFast, the ActvSAMP process will select similar users from the same cluster, and so the model performance will remain unaffected by periods of user dropout. A more detailed analysis of this scenario is part of a future work.

For future work, we will also focus on improving our solution to actively minimise the number of clients needed to train the model to further reduce the communication-related battery consumption.

Although FedFast learns effective user and item representations, it is still susceptible to the cold start problem since it expects that the number of users and items are provided during training. The model has to be retrained, albeit cheaper and faster, to support new users and items — through transfer learning, the process can reuse the pre-trained weights of the user and item embeddings.

Finally, while federated learning methods, e.g. FedFast, are vulnerable to model inversion attacks, this vulnerability can be mitigated through *differential privacy* and other privacy enhancing

techniques. Discussing the privacy implications of FL is beyond the scope of this work and will be investigated in future work.

6 CONCLUSION

We propose FedFast, a method for making accurate distributed recommendations using deep neural networks and federated learning. Without compromising on the model's recommendation accuracy, FedFast uses a novel sampling technique and a novel aggregation strategy to demonstrably improve the convergence speed required to train a federated recommendation model, by exploiting user profile similarities. We evaluated our technique against standard centralised and federated learning baselines using four real-world datasets. Our results show that FedFast outperforms the federated learning baselines in terms of convergence speed and accuracy. FedFast reaches the maximum baseline NDCG at least 4 times faster (in ML100K) than the baseline and in some cases more than 20 times faster (in TripAdvisor). We are also able to get similar recommendation performance as the centrally trained model and at most cases comparable to BPR. Future work will consider new approaches to make the training process even more efficient by modelling the sampling of users as a reinforcement learning process, for example. We will also consider more sophisticated models that improve the accuracy of the underlying recommendation model, and by extension the its federated version. We will also work on techniques to minimise the number of users that participate in the training process without affecting the performance of the model.

ACKNOWLEDGMENTS

This work was supported by Samsung Research, Samsung Electronics Co. Ltd, and Science Foundation Ireland through the Insight Centre for Data Analytics under grant number SFI/12/RC/2289_P2.

REFERENCES

[1] Marharyta Aleksandrova, Armelle Brun, Anne Boyer, and Oleg Chertov. 2017. Identifying Representative Users in Matrix Factorization-based Recommender Systems: Application to Solving the Content-less New Item Cold-Start Problem. *Journal of Intelligent Information Systems* 48, 2 (2017), 365–397.

[2] Muhammad Ammad-ud-din, Elena Ivannikova, Suleiman A. Khan, Were Oyomno, Qiang Fu, Kuan Eeik Tan, and Adrian Flanagan. 2019. Federated Collaborative Filtering for Privacy-Preserving Personalized Recommendation System. *CoRR* abs/1901.09888 (2019), 1–12. arXiv:1901.09888

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. *CoRR* abs/1902.01046 (2019), 1–15. arXiv:1902.01046

[4] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. 2018. Federated Meta-Learning with Fast Convergence and Efficient Communication. arXiv:cs.LG/1802.07876

[5] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. 2017. On Sampling Strategies for Neural Network-based Collaborative Filtering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, USA, 767–776.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, New York, USA, 191–198.

[7] Mukund Deshpande and George Karypis. 2004. Item-based Top-*N* Recommendation Algorithms. *ACM Transactions on Information Systems* 22, 1 (2004), 143–177.

[8] Ruihai Dong, Michael P O'Mahony, Markus Schaal, Kevin McCarthy, and Barry Smyth. 2016. Combining Similarity and Sentiment in Opinion Mining for Product Recommendation. *Journal of Intelligent Information Systems* 46, 2 (2016), 285–312.

[9] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. 2015. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 278–288.

[10] Jack Goetz, Kshitiz Malik, Duc Bui, Seungwhan Moon, Honglei Liu, and Anuj Kumar. 2019. Active Federated Learning. arXiv:1909.12641

[11] Joshua Grass and Shlomo Zilberstein. 1996. Anytime Algorithm Development Tools. *ACM SIGART Bulletin* 7, 2 (Apr 1996), 20–27.

[12] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. 2018. Outer Product-based Neural Collaborative Filtering. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, Vol. abs/1808.03912. IJCAI, Stockholm, Sweden, 2227–2233.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, International World Wide Web Conferences Steering Committee, Geneva, Switzerland, 173–182.

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:cs.CV/1704.04861

[15] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:cs.LG/1610.05492

[16] Yehuda Koren. 2008. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, ACM, New York, USA, 426–434.

[17] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (Aug 2009), 30–37.

[18] Bin Li, Qiang Yang, and Xiangyang Xue. 2009. Transfer Learning for Collaborative Filtering via a Rating-matrix Generative Model. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, New York, USA, 617–624.

[19] Nathan N. Liu, Xiangrui Meng, Chao Liu, and Qiang Yang. 2011. Wisdom of the Better Few: Cold Start Recommendation via Representative Based Rating Elicitation. In *Proceedings of the Fifth ACM Conference on Recommender Systems*. ACM, ACM, New York, USA, 37–44.

[20] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.), Vol. 54. PMLR, Fort Lauderdale, USA, 1273–1282.

[21] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. *2019 IEEE Symposium on Security and Privacy* 1, 1 (May 2019), 1–15.

[22] Weike Pan, Evan Wei Xiang, Nathan Nan Liu, and Qiang Yang. 2010. Transfer Learning in Collaborative Filtering for Sparsity Reduction. In *Proceedings of the 24th Conference on Artificial Intelligence*. AAAI Press, Atlanta, Georgia, 230–234.

[23] Dan Pelleg and Andrew W. Moore. 1999. Accelerating Exact *k*-means Algorithms with Geometric Reasoning. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, USA, 277–281.

[24] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Montreal, Canada, 452–461.

[25] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of the 24th International Conference on World Wide Web*. Association for Computing Machinery, New York, USA, 111–112.

[26] Lei Shi, Wayne Xin Zhao, and Yi-Dong Shen. 2017. Local Representative-Based Matrix Factorization for Cold-Start Recommendation. *ACM Transactions on Information Systems* 36, 2 (2017), 22:1–22:28.

[27] Bo Zhang, Na Wang, and Hongxia Jin. 2014. Privacy Concerns in Online Recommender Systems: Influences of Control and User Data Input. In *10th Symposium On Usable Privacy and Security ({SOUPS} 2014)*. USENIX Association, Menlo Park, CA, 159–173.

[28] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys (CSUR)* 52, 1, Article 5 (Feb 2019), 38 pages.

[29] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-N Collaborative Filtering via Dynamic Negative Item Sampling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, USA, 785–788.