arXiv:2309.08635v1 [cs.IR] 14 Sep 2023

FedFNN: Faster Training Convergence Through Update Predictions in Federated Recommender Systems

Francesco Fabbri[1], Xianghang Liu[2], Jack R. McKenzie[2], Bartlomiej Twardowski[3], and Tri Kurniawan Wijaya[3]

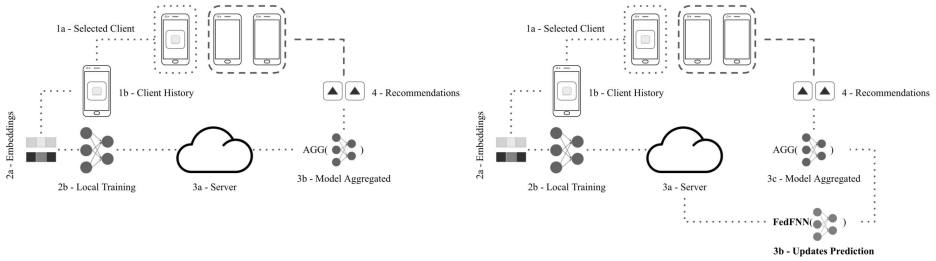
[1] [1] Eurecat, Barcelona, Spain
francesco.fabbri@eurecat.org
[2] [2] Huawei UK Research Centre, London, UK
xianghang.lui@huawei.com
jack.mckenzie1@huawei.com
[3] [3] Huawei Ireland Research Centre, Dublin, Ireland
bartlomiej.twardowski@huawei.com
tri.kurniawan.wijaya@huawei.com

Abstract. Federated Learning (FL) has emerged as a key approach for distributed machine learning, enhancing online personalization while ensuring user data privacy. Instead of sending private data to a central server as in traditional approaches, FL decentralizes computations: devices train locally and share updates with a global server. A primary challenge in this setting is achieving fast and accurate model training—vital for recommendation systems where delays can compromise user engagement. This paper introduces FedFNN, an algorithm that accelerates decentralized model training. In FL, only a subset of users are involved in each training epoch. FedFNN employs supervised learning to predict weight updates from unsampled users, using updates from the sampled set. Our evaluations, using real and synthetic data, show: (*i*) FedFNN achieves training speeds 5x faster than leading methods, maintaining or improving accuracy; (*ii*) the algorithm's performance is consistent regardless of client cluster variations; (*iii*) FedFNN outperforms other methods in scenarios with limited client availability, converging more quickly.

1 Introduction

Federated Learning (FL) has become a pivotal formulation for distributing the computation of machine learning (ML) models. This approach improves online services, such as content personalization, while simultaneously preserving user privacy. This paradigm is gaining significant traction, particularly for use cases like next-word prediction or item suggestions. This is because a personalized output can be generated on-device (e.g., smartphone, tablet), negating the need to transfer sensitive data to an external server [3, 17, 24, 26].

Unlike traditional centralized learning approaches where private user data is sent to a central server for synchronous training, an FL setting distributes



(a) FedAvg (b) FedFNN

Fig. 1: On the upper figure (a), an illustration of the Federated Learning training process: (1) the interactions of the client with the device are converted to vector embeddings (for both items and users); (2) then the local training on the selected devices; (3) the new training information are then sent to the server to be aggregated. (4) the server sends the updates to all the clients which can generate the recommendations. On (b) the introduction of FedFNN, which generates the predictions of the unselected clients before sending the new updates.

computations across multiple devices. A server coordinates the local training of devices and aggregates their updates. This structure mitigates privacy risks like personal data leakage, misuse, or abuse since client training data remains undisclosed to the central server.

In the FL framework, mobile devices (**clients**) train their ML model locally, sharing only model updates—not the raw historical data—with a centralized unique aggregator (**server**) [18, 7, 17, 11]. The server coordinates training and distributes model updates to clients at each round.

In this work, our focus is on applying FL to recommender systems. These systems are trained on user history data and predict users' preferences for items. Both user and item data are typically represented through a low-dimensional embedding, a critical component for most state-of-the-art recommendation models [19, 27].

The training mechanism of FL models is delineated in Figure 1a. Initially, a subset of clients is chosen. The client's historical interactions (denoted by 1b - Client History) are then transformed into embeddings (indicated as 2a - Embeddings). These embeddings undergo local training (2b - Local Training) and are subsequently updated. Post this local training phase, the embedding alterations are transmitted to the server (3a - Server). The server aggregates these updates (3b - Model Aggregated) and relays the new data to all clients, enabling them to generate recommendations (4 - Recommendations).

This pipeline, proposed by [17] represents the standard for training FL models in which, the representations for the trained users and items will be aggregated as an average on the server, before then being redistributed among untrained clients.

FL training can occasionally be hindered by specific data characteristics, leading to protracted training durations and sub-optimal model performance. Notable among these are: (*i*) *sparsity* - a scenario where users typically engage with only a limited array of items; (*ii*) *heterogeneity* - wherein each client possesses data pertaining exclusively to its user; and (*iii*) *poor client availability* - during each training round, only a subset of clients (referred to as **delegates**) participate, leaving a significant majority (**subordinates**) inactive.

To circumvent these challenges and expedite training, we introduce `FedFNN`. This innovative algorithm harnesses a meta-network to learn and predict the embedding updates from local client training. This meta-network can proactively update embeddings even for clients that weren't chosen in the initial selection, thus accelerating convergence, as visualized in Figure 1b.

Our model, `FedFNN`, amalgamates recent advancements targeting training speed optimization [18]. It has demonstrated superiority in both convergence speed and prediction accuracy. The model utilizes a supervised learning approach to anticipate updates for unseen clients during each training iteration. Its efficacy has been empirically validated on both synthetic and real-world data sets. Our evaluations pitted `FedFNN` against the competitive `FedFast` algorithm and various other baselines. Results showcased `FedFNN`'s prowess, as it consistently delivered results at least five times faster in terms of the number of iterations required to achieve comparable accuracy levels.

To simulate scenarios of low client availability and data sparsity, we employed a synthetic data generator. This tool facilitates the artificial generation of user-item interaction data, allowing the creation of distinct client clusters based on preference. Our findings underscore the robustness of `FedFNN`, both in terms of convergence speed and prediction accuracy, regardless of the client group size or availability.

Additionally, we conducted an ablation study to decipher the contribution of various `FedFNN` components. Specifically, we delved into the realms of item update aggregation and subordinate predictions. Our investigations revealed that our method for predicting subordinates emerged as the most optimal, regardless of the item update strategy employed.

Paper Structure. The subsequent sections of this paper are organized as follows: Section 2 delves into relevant literature. In Section 3, we elucidate the background and provide a formal description of our algorithm. Section 5 details our experimental design and presents the results. Concluding remarks and potential avenues for future research are discussed in Section 6

2 Related Work

Federated Learning has emerged recently as popular research area, since it allows exploitation of the vast amount of data collected on personal devices through privacy-preserving technologies [7, 11]. [17] was the first to introduce the definition of Federated Learning, with the Federated Averaging (`FedAvg`) algorithm

that is capable of increasing the training speed of classification models through the sharing of weight updates among sampled clients [17].

Parallel to the `FedAvg` improvements, there is a new line of work proposing increased accuracy of models in the presence of heterogeneous data distributions among devices (i.e. an imbalanced distribution of samples among the different devices), also called *statistical heterogeneity* [12, 20, 9]. [12] propose `FedProx`, which modifies `FedAvg` algorithm by including a proximal term, reducing differences between local and global updates [12].

The FL paradigm has been mainly investigated for supervised learning tasks (e.g. binary and multi-label classification). Due to its effectiveness of generating on-device personalised recommendations without violating the user privacy, it has also been applied to the area of information retrieval and recommendation algorithms [22, 18, 1, 8, 13].

Most of the contributions focus on the matrix factorisation algorithm, which represents the standard to capture latent dimensions of user-item interactions [6, 10]. [1] introduce `FedeRank`, a federated recommendation algorithm which allows users to control the portion of data they want to share [1]; it focuses on the accessibility of personal data, which is an aspect not covered by `FedFNN`. [14] propose `FedRec`, a federated model for predicting explicit feedback [14]; in contrast, our work focuses on implicit feedback. Additionally, their objective is to generate performances as close as possible to the centralised version of the model, whereas our focus is to improve training speed.

[18] propose `FedFast` algorithm that proposes to to speed up the recommender system training by clustering the users [18]. Specifically, they propose: (*i*) a model to update unsampled clients, (*ii*) an item update strategy which looks at client contribution weighting and (*iii*) a sampling technique that is coherent with the clustering approach. Compared to our work, they rely on the assumption of having clustered embeddings in both, the input data distribution and the clients sampled for the local update. We show through an ablation study how their clustering approach does not significantly help in the prediction task, presenting performances close to the baseline (with no prediction for the unsampled clients). Our work aims to capture similarities at individual and not group level, between user embeddings. We extensively compare `FedFNN` to `FedFast` in Section 5, showing the effectiveness of our solution.

`FedFNN` intuition is built on the idea of predicting differences in the next weights updates of the subordinate clients, which intuitively can be connected to [5], a popular architecture achieving state-of-the-art performances in Computer Vision tasks, where the hidden layers are enhanced with the prediction of the difference between the new and old weights. [4] proposes hypernetworks, strongly related to our work, where a single network is used to generate the weights of another architecture [4, 25]. In hypernetworks, one architecture is trained and the other predicted, while in our case, `FedFNN` is trained to predict the updates of the weights of the main model, which then uses the updates for the local training. Another relevant line of research connected to hypernetworks is meta-learning, which has been also explored in connection with Federated Learning [15, 2].

Specifically, in those approaches a meta recommender system is designed, able to generate item embeddings with a meta network.

`FedFNN` is built on Matrix Factorisation algorithm, which represents the standard for recommender systems in presence of implicit feedback [10]. However recently, models more sophisticated than that have also been proposed. Among many, [23] introduced a federated framework for online advertising to predict CTRs of native ads from multi-platform user behavior [23], while [22] introduce a graph neural network architecture to predict high-order user interactions [22]. `FedFNN` can be extended to other sophisticated models in a FL setting and we leave this task for the future work.

3 Preliminaries

Federated Learning is characterised by a unique server which handles incoming messages from multiple clients. The server aggregates these messages to optimise a single global optimum. In particular the objective is to minimise:

$$f(v, w) = \min \sum_{k=1}^{N} p_k F_k(v, w) \tag{1}$$

where N is the number of devices, v the item embeddings, w the user embeddings, $F_k(v, w)$ the local objective for the device k, which measures the local empirical risk over data distribution D_k, and the normalisation factor $p_k \in [0, 1]$ applied to $F_k(v, w)$. Let the learnable parameters of the General Matrix Factorisation (`GMF`) algorithm be denoted by $\{v, w, \phi\}$. The parameter set ϕ encompasses all the additional parameters intrinsic to the `GMF` algorithm, excluding v and w These parameters, although less emphasized in our context, are indispensable for the effective training of the `GMF`.

This model is widely used in designing recommendation algorithms and predicts user preferences through a low-dimension representation of the data.

To learn the user preferences from the embeddings, it uses a fully connected layer that takes as input user and item embeddings and outputs the scores distribution for all the missing interactions [6].

This model can then be naturally extended in a federated setting using `FedAvg`, where a subset of all devices are selected for local updates. Algorithm 1 presents the steps of `FedAvg`; each k-th sampled client (line 3) gets first a copy of the model to train on its own local data D_k, and subsequently the loss function L optimised by the local model with respect to $\{v_k, w_k, \phi_k\}$ (lines 4,5):

$$\{v_k, w_k, \phi_k\} \leftarrow \mathbf{A}\ (v, w, \phi, L, D_k), \text{ for each } k \in S, \tag{2}$$

where (v_k, w_k, ϕ_k) are the learnable parameters updated by the delegate client k, $\mathbf{A}$ is the optimisation algorithm selected to estimate the parameters values and $S \subseteq \{1, \ldots, N\}$ is the indices of all the delegates. Since we work with implicit feedback data, a binary cross-entropy loss function is selected; for the optimisation algorithm the ADAM optimiser is chosen.

Algorithm 1: FL Algorithm

Input : set of item I, set of clients U, sample size m

1 Initialise: user embeddings w^0, item embeddings v^0;
2 **for** $t \leftarrow 1, 2, \ldots, T$ **do**
3 $S_t \leftarrow$ ClientSampling(U, m);
4 **for** *client* $k \in S_t$ **do**
5 $w_k^{t+1}, v_k^{t+1} \leftarrow$ LocalUpdate(w_k^t, v_k^t, k);
6 $\Omega_t \leftarrow w^{t+1}, S_t, m$;
7 **for** *client* $j \in U \backslash S_t$ **do**
8 $w_j^{t+1} \leftarrow$ ClientPrediction(Ω_t);
9 $v^{t+1} \leftarrow$ ItemUpdate($v^t[S_t]$);

For each training round, the weights updated through the local models are aggregated to generate the global update (lines 6-9). In particular, among all the client embeddings included in the global model $w[D]$, when using `FedAvg`, only the embeddings of the delegates S (i.e. $w_k[k], k \in S$) will be updated:

$$w_k^{new}[k] \leftarrow w_k[k], \text{ for each } k \in S \tag{3}$$

In contrast, the item embeddings are aggregated in the server (lines 9-10) and then redistributed among all the clients.

Updating only a subset of clients represents a limitation that affects the number of rounds necessary to get good model performances. In this direction, subordinate clients updates are crucial to the convergence of the global model. In `FedAvg` the embeddings of the local clients are updated as follows:

$$w^{new}[j] \leftarrow \frac{1}{|S|} \sum_{k \in S} w_k[k], \text{ for each } j \in U \setminus S \tag{4}$$

To update embeddings of subordinate clients, `FedFast` recently proposes a clustering-based model aggregation approach to update the embeddings of subordinate users with the averaged weight update of the selected users belonging to the same cluster [18]. First, users are clustered based on their embeddings $C = \{c_i \mid i = 1, 2, \ldots\}$, where each cluster c_i is a set of users. Then, the averaged weight update δ_{c_i} is computed for all clients in c_i:

$$\delta_{c_i} \leftarrow \frac{1}{|c_i \cap S|} \sum_{k \in c_i \cap S} w_k[k] - w[k], \text{ where } i = 1, 2, \ldots, |C| \tag{5}$$

The new averaged update is used for modifying the weights of the subordinate users:

$$w^{new}[k] \leftarrow w[k] + \delta_{c_i}, \text{ for each } k \in c_i \setminus S \tag{6}$$

Despite being faster to converge than `FedAvg`, `FedFast` relies on two assumptions which may be difficult to maintain consistently: *(i)* it assumes that clients

can be clustered into distinct groups, which is not always the case (e.g. group of users presenting highly overlapping preferences); *(ii)* even upon assuming the existence of the groups of preferences, fixing an *a priori* number of clusters for the sampled clients relies on the feasibility to sample users from all the clusters at each round. Moreover, since real data cannot be accessed in a unique pass, finding the correct number of clusters in the federated setting may require multiple tests before the final model training. Due to these limitations, as is found in our ablation study in Section 5, their cluster-based update propagation produces little to no improvements over the training of the models.

4 FedFNN

To overcome the limitations mentioned above, we relax the assumption of the clustered preferences to the assumption that if two users are close in the embedding space, their embeddings would have similar updates after local learning. Under this new, relaxed assumption, we propose `FedFNN` that directly predicts the embedding weight updates for subordinates (i.e. users that are not directly sampled and updated in one epoch) using supervised regression models. This idea is also closely related to the class of meta-learning algorithms which directly generates the weights of the target model [4, 25]. However, instead of directly generating the embedding weights, `FedFNN` predicts the embedding updates using those produced by the delegate clients.

The training phase is performed on the data coming from the delegates to optimise a learning function $f(\cdot;\theta)$, using the previous user embeddings as input variable and the new embeddings updates as output:

$$\min_{\theta} : \mathbb{E}_U \left[\ell \left(f \left(w[U]; \theta \right), w^{new}[U] - w[U] \right) \right] \tag{7}$$

The objective function is the expected RMSE taken over the distribution of all the clients U as we are interested in the generalisation error of the update predictor. The architecture for $f(\cdot;\theta)$ proposed in the experiments with FedFNN is a multi-layer perceptron (MLP), which proves to be simple but effective with the data sets we use.

Then, for each subordinate client we predict the updates

$$w^{new}[j] \leftarrow e^{-\gamma t} \ell(w[j]; \theta^*) + (1 - e^{-\gamma t}) w[j], \text{ for each } j \in U \setminus S \tag{8}$$

where γ is the discount factor and t the number of running epochs. Both of these parameters are combined to generate an exponential decay and reduces the contribution of the new predictions after each iteration. The updates are predicted by minimising the error produced on average (RMSE) and since weights need to converge towards personalised values dependent on the client information, those predictions becoming less relevant in the long-run.

In contrast to `FedFast`, which uses unsupervised learning (e.g., k-means), `FedFNN` uses supervised learning that explicitly models the mapping from the

Algorithm 2: FEDFNN

Input : $\Omega_t \leftarrow \{W_t, S, W_{t+1}[S]\}$, prediction function $g(.)$, number of delegates m, patience criteria p
Output : $W_{t+1}[D]$, $RMSE_t$

if $t \leq p$ ***or*** $|1 - L(t)/L(t-p)| \geq \epsilon$ **then**
  $\delta[S] \leftarrow W_{t+1}[S] - W_t[S]$;
  $\theta^*, RMSE_t \leftarrow \arg\ \text{opt}_\theta \sum_{k \in S} \ell(g(w_t[k]; \theta), \delta[k])$;
  for *client* $j \in D \backslash S_t$ **do**
    $w_{t+1}[j] \leftarrow e^{-\gamma t} g(w_t[j]; \theta^*) + (1 - e^{-\gamma t}) w_t[j]$;

client representation to the user embedding updates. Additionally, `FedFNN` enables the introduction of more sophisticated machine learning models (e.g. Neural Networks), adding the ability to capture more information and increasing modeling power. Later, we show through an ablation study that updating the subordinate client embedding using `FedFNN` can generate significant improvements over the training of the models, which is not the case for the clustering method.

4.1 Optimisation

To find the optimal parameter θ in Eq. 7, we adopt a standard supervised learning approach based on cross validation. More specifically, for each communication round, we run 5-fold cross validation on the delegate clients, train $f(\cdot; \theta)$ on the training set using SGD based optimisation methods and approximate the objective in Eq. 7 with the empirical loss on the validation set. During each round, we use grid-search to optimise the hyper-parameters on the validation set. Our set of hyper-parameters include: the network architectures, learning rates and dropout ratios for regularisation. Moreover, since FedFNN is designed to optimise the training of the global model at early stage of the iterations, we use a *patience* parameter p, similar to the early-stopping criteria used to avoid over fitting in supervised learning [16], to stop predicting the subordinates embeddings. Specifically, at each iteration t we monitor the loss function value of the global model with respect to the value at iteration $(t-p)$. Once the loss function stabilizes around a previous value, we stop using FedFNN. More precisely, we stop using FedFNN when $|1 - L(t)/L(t-p)| < \epsilon$ (where $\epsilon = 0.01$), where L represents the loss function of the global model. Algorithm 2 presents the whole process.

For the item updates, various strategies can be applied to define the importance of the local update generated through the client k. Specifically, a coefficient α_k can be defined for each delegate client. In particular we implement `FedFNN` along with 3 different strategies:

- **W0**: The new local embedding updates are re-distributed through averaging the new changes among all the sampled items, where $\alpha_k = 1/m$, $\forall k \in S$

– **W1**: The local item updates are distributed following the weighted average proposed in [18], where the importance of weights updates in the current iteration is distributed according to the difference between the update and the embedding. Let

$$Z_{t+1}[k] = \sum_{v_i \in D_k} ||v^i_{t+1}[k] - v^i_t||_1. \tag{9}$$

Then, we set

$$\alpha_k = \frac{Z_{t+1}[k]}{\sum_{k \in S_t} Z_{t+1}[k]}. \tag{10}$$

In this way, the clients presenting bigger updates are valued more.
– **W2**: The local item updates are weighted by the number of samples included in each delegate client, where $\alpha_k = \frac{n_k}{\sum_k n_k}$, where n_k is the number of interactions present in the client history.

4.2 Sampling

In contrast to the `FedFast` algorithm, we do not add any constraint to the sampling strategy. We assume that in a practical, real use case scenario, sampling would be determined via external factors (e.g. client availability, connection quality, etc), not the algorithm being used.

5 Experiments

This section focuses on answering the following research questions: ($RQ1$) What is the performance of `FedFNN` compared to the state-of-the-art in terms of accuracy and speed of convergence?; ($RQ2$) Do the different strategies for updating both user and item embeddings significantly increase the speed of convergence for FL models?; ($RQ3$) How might distribution of user preferences affect the performances of `FedFNN`?; ($RQ4$) How might client availability and sample size impact the performances of `FedFNN`?

Data. For the experimental section we use 3 different data sets: MovieLens, a popular movies reviews' collection used for bench-marking recommendation algorithms (**ML-100K**); Yelp, an aggregator of shops reviews (**YELP**); Amazon Music sample of songs reviews (**AMAZON**). The characteristics of the 3 data sets are summarised in Table 1.

data set	Interactions	Users	Items	Sparsity
AMAZON	64,706	5,541	3,568	0.0135
ML-100K	100,000	943	1,682	0.0630
YELP	267,211	16,442	11,128	0.0015

Table 1: Summary of data used for the experiments

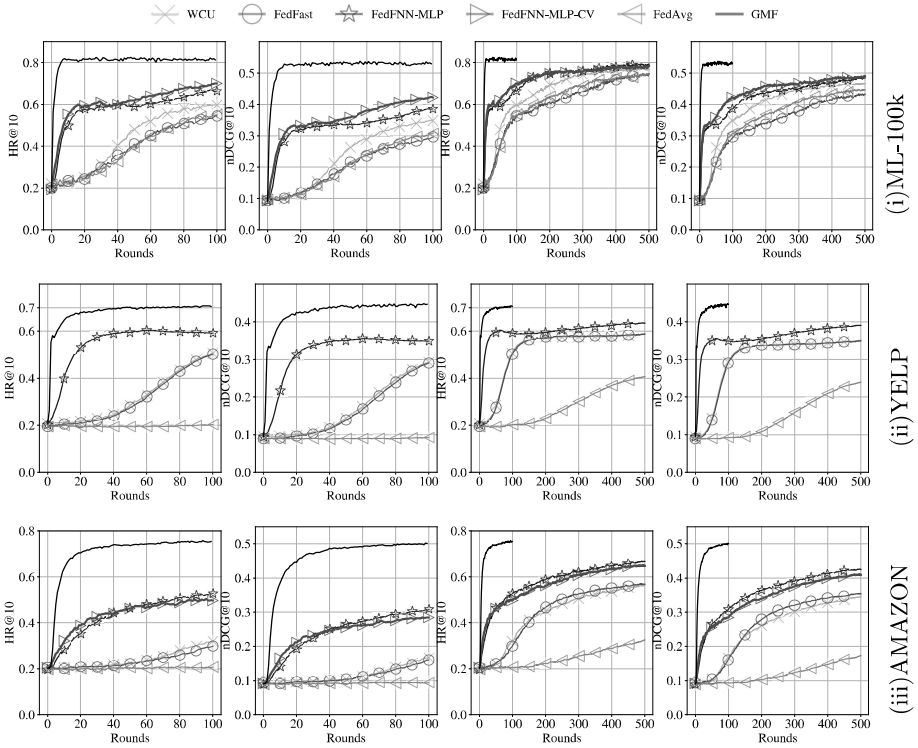


Fig. 2: Comparison of FL models on three different data sets. The first two columns show only the first 100 iterations, while the third and fourth columns are showing up to 500 iterations.

Algorithms. We test `FedFNN` with state-of-the-art algorithms and a baseline: (*i*) `FedFast`, the current state-of-the-art, in terms of speed, to the best of our knowledge. It predicts the subordinates through a clustering based approach ([18]); (*ii*)`FedAvg`, baseline on which `FedFNN` and `FedFast` have been designed. It distributes local updates (both item and client embeddings) through averaging, and does not predict embeddings of the subordinate clients ([17]); (*iii*)`WCU`, *Without Client Updates* (WCU) algorithm only updates the embeddings of the sampled clients and the items that have been interacted with; subordinate embeddings are not updated. We include this option to emphasise the differences in performances with `FedFast`, which the only difference is the inclusion of the clustering method for predicting the unsampled clients; (*iv*) `GMF`, the centralised version of the Neural Matrix Factorisation algorithm. It is included to establish the upper-bound of the FL-based solutions [6]; (*v*) `FedFNN-MLP`, `FedFNN` implementation using a multi-layer perceptron (MLP) to predict the unsampled clients; (*vi*) `FedFNN-MLP-CV`, the architecture is the same of FedFNN-MLP, but this time the hyperparameters of `FedFNN` (dropout, number of hidden neurons,

number of layers) are optimised running a grid-search in a 5-fold cross validation, at each training round.

For the item embedding updates it is important that an optimal strategy for redistributing the new item updates ($v^{t+1}[k]$) through a weighted average is found. For this reason we test different item embedding updates (**W0, W1** and **W2**).

5.1 Performance and Accuracy (RQ1, RQ3)

Accuracy and Convergence Comparison. Here we compare different `FedFNN` implementations with the algorithms and data sets introduced above. In Figure 2 we show the results generated after both 100 and 500 iterations, where the first two columns of plots show the first 100 iterations (HR@10 and nDCG@10), while the second and third columns show the whole training rounds. In order to generate a fair comparison: (*i*) for each data set, the learning rate of the local updates is the same for all the algorithms; (*ii*) W1 is the item embedding strategy used for `FedFast`, `FedFNN` and `WCU` implementations; (*iii*) the sample size of the local updates is fixed to 10% for all the experiments.

Overall, `FedFNN-MLP` consistently outperforms the other algorithms on all 3 data sets, both in terms of speed of convergence (the number of iterations needed to reach stable level of convergence), and final accuracy (the final NDCG and HR). In our experiments we focus on the first 100 iterations of training to check the speed of convergence during this early stage: with ML-100K data, `FedFNN-MLP` is 2x faster than `FedFast`, for YELP data it is 5x faster, and for AMAZON it is even 8x faster than `FedFast`. Specifically, `FedFNN-MLP` needs only 12 iterations to reach `FedFast` accuracy at the 100th iteration. On the other hand, `FedFast` is better than `FedAvg` in terms of performance and accuracy, but presents similar performances of `WCU`, which is characterised by the absence of clustering techniques. This means that the clustering approach used to predict the clients' embeddings appears to have little effect on the performance of the algorithm. The introduction of the cross validation for `FedFNN`, helps to generate robust predictions (`FedFNN-MLP-CV`), presenting an increase in performance in the early stage of training, for all the experiments. In the case of MovieLens data set, `FedFNN-MLP-CV` is also slightly improving the overall performance of the standard `FedFNN-MLP`.

Grouped Preferences. In order to analyse the performance of `FedFNN` in presence of clustered preferences we design a synthetic data generator which allows to input the number of groups of users presenting same preferences when generating the user-item interactions. We can first indicate the desired number of users (N) and items (M), then a mapping function $\ell : D \rightarrow \{D_1, D_2, ...D_G\}$, which maps each user-item interaction to a different subset D_j, where G is the total number of groups in input. For sake of simplicity, group of users do not overlap in terms of item preferences an all the groups have the same relative size. To generate more realistic data, we include parameters to model sparsity and popularity as a power-law distribution, which is generated through a *Beta(a,b)* random variable.

First, we use the Beta (with parameters fixed to $a = 1, b = 3$) for generating the deviation from the average s for all users, then, we assign to each user u the final level of sparsity $s_u = s + s(s_u - s_M)$, where s_M is the median of the sparsity vector generated through the sampling. The *Beta* parameters are chosen to lead to a level of sparsity similar to the one in the real data set introduced in Table 1. For each group of items D_j, we can define a different popularity bias vector, $\mathbf{p}_j$ which for each item i in the group j, maps a coefficient $p_j(i) \in [0, 1]$ representing the chances of being selected. In this way, when generating user preferences, first the number of interactions is computed ($n_u = \lceil M \times s_u \rceil$) and then n_u items are sampled following $\mathbf{p}_j$. The detailed process follows: (1) we initialise the number of users N, the number of items M, the sparsity value s and the number of groups G. Initialise the sparsity sample and generate the sparsity value for each user (s_u); (2) for each group we generate the popularity vector $\mathbf{p}_j$, sampling from the beta distribution; (3) for each user we then sample n_u items following the probability:

$$p(u, i) = \frac{h(u, i)p_i}{\sum_{j \in I} h(u, j)p_j}$$

where $h(u, i) = \eta$ if u and i are mapped through $\ell(.)$ to the same group, otherwise $h(u, i) = (1 - \eta)$. This parameter introduces a certain level of randomness, to have more realistic clusters distribution. In our experiments, we fix it to $\eta = 0.9$.

We test `FedFNN` and `FedFast` on 6 different data sets, which differ only by the number of input clusters, where $G \in \{5, 10, 50, 100, 500, 1000\}$. In this case, `FedFast` is trained given in input the exact number of clusters used to generate the data. Also the parameters used to generate sparsity and popularity bias are the same along all the data sets. The sample size and local learning rate for the training of the algorithm does not change for the experiments. The results are presented in the Figure3, where the metric used is the difference in performance between `FedFNN` and `FedFast`, i.e. Δ_{HR} and Δ_{nDCG}, positive if `FedFNN` presents better accuracy, negative if `FedFast` is more accurate. This value is measured at three different moments along the training: at early stage (after 100 iters); in the middle (after 300 iterations) and at the end of the training (after 500 iterations). It is possible to highlight how `FedFNN` is consistently better in terms of accuracy, looking at the values of Δ_{HR} and Δ_{nDCG}. The improvement of `FedFNN` over `FedFast` is most noticeable in the early stages of learning (at 100 iterations), but even at later stages (after 300 and 500 iterations) it is consistently better in terms of accuracy. Since through the local updates, the two algorithms can only access to a portion of the whole data set, the clustering approach (`FedFast`) may struggle to sample necessary information to properly assign the clients to the right clusters.

Model	100 iter HR@10	100 iter nDCG@10	300 iter HR@10	300 iter nDCG@10	500 iter HR@10	500 iter nDCG@10
ML-100k						
W0+WCU	0.4740	0.2636	0.6352	0.3859	0.7137	0.4331
W0+FedFast	0.4995	0.2826	0.6691	0.4060	0.7158	0.4299
W0+FedFNN	**0.5801**	**0.3421**	**0.6893**	**0.4180**	0.7063	**0.4365**
W1+WCU	0.5981	0.3487	0.7444	0.4445	0.7678	0.4666
W1+FedFast	0.5790	0.3255	0.7264	0.4435	0.7614	0.4620
W1+FedFNN	**0.7084**	**0.4281**	**0.7667**	**0.4755**	**0.7890**	**0.4942**
W2+WCU	0.5451	0.3050	0.7137	0.4262	0.7561	0.4648
W2+FedFast	0.5323	0.2904	0.7010	0.4148	0.7529	0.4551
W2+FedFNN	**0.6225**	**0.3616**	**0.7222**	**0.4425**	**0.7614**	**0.4768**
Yelp						
W0+WCU	0.1999	0.0916	0.2013	0.0921	0.2048	0.0952
W0+FedFast	0.2030	0.0937	0.2101	0.0977	0.2163	0.1021
W0+FedFNN	**0.2147**	**0.0986**	0.2094	0.0975	**0.2241**	**0.1058**
W1+WCU	0.5042	0.2918	0.5768	0.3404	0.5880	0.3513
W1+FedFast	0.5017	0.2895	0.5774	0.3403	0.5876	0.3491
W1+FedFNN	**0.5910**	**0.3488**	**0.6123**	**0.3703**	**0.6342**	**0.3902**
W2+WCU	0.2036	0.0927	0.2890	0.1506	0.3963	0.2319
W2+FedFast	0.2053	0.0937	0.2729	0.1373	0.3895	0.2262
W2+FedFNN	**0.3099**	**0.1636**	**0.4082**	**0.2381**	**0.4385**	**0.2606**

Table 2: Summary of the ablation study conducted to analyse the impact of user and item embedding updates. In bold, the model presenting the best performances by different item embedding updates.

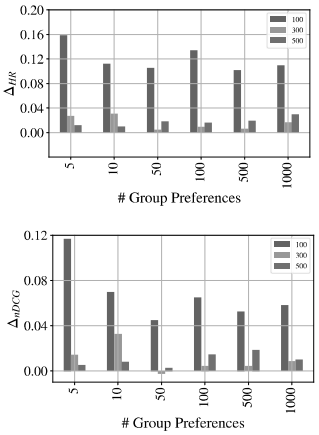


Fig. 3: Comparsion of FedFast and `FedFNN` performances when users present different grouped preferences.

5.2 Ablation Study (RQ2)

In order to study the contribution of the algorithm proposed, we perform an ablation study to emphasise the importance of using different strategies for both client and item embedding updates. For this reason we test three different strategies of subordinates predictions (`FedFNN`, `FedFast` and `WCU`) and three different strategies of item embedding updates (W0, W1 and W2). The experiments are performed following the same configurations used in Figure 2. In Table 5.1 we present the results for MovieLens and Yelp data sets, highlighting accuracy (HR@10 and nDCG @10) after 100, 300 and 500 iterations. About item embedding updates, for both data sets, **W1** presents better performances, in terms of speed and accuracy. Moreover, W2 presents better performances when compared to W0, which is consistently the slower one, no matter the choice of client predictions strategy. Among the client updates, `FedFNN` is optimal in terms of accuracy and speed of convergence, for both data sets, at all stages of training. `FedFast` and `WCU` demonstrate similar performance.

5.3 Poor availability and Sampling (RQ4)

We explore two different scenarios which are of practical significance in a FL setting: (*i*) limited availability of clients, where along the training a group of client is consistently less available to be sampled (e.g. clients penalised when sampling the subset for the local updates); (*ii*) a smaller fraction of clients are sampled to participate in training.

Availability. Not all clients will have the same levels of availability during training, and those who participate less in training will have a diminished performing model. It is for this reason we decided to test how clients who participate less respond to the `FedAvg`, `FedFast` and `FedFNN` algorithms. Using ML-100K data set, we split (uniformly at random) the clients in two blocks of same size. One block is fully available along the training (**normal availability**), while all the clients belonging to the other have 25% chance of being available at each iteration for being sampled and run the local update (**poor availability**). Partitioning the clients this way means that the clients assigned to the poor availability group are consistently sampled and locally updated less frequently than the other group. We test and monitor the performances of three different models in this case (`FedFNN`, `FedFast` and `FedAvg`). In Figure 4b we report on the left (Fig. 4a) the performance for the portion of users consistently available, while on the right (Fig. 4b) the performance of the sample *poorly available*. `FedFNN` shows better performances for the both groups of clients. In particular, the gap in terms of accuracy is larger for the poor availability clients, especially at the early stages of the training. For both groups of clients, `FedFast` and `FedFNN` stabilize around the same level of accuracy, while `FedAvg` fails to reach the same level of accuracy for the poor availability group.
Sampling. In this experiment we explore how different sampling rates affect the convergence of `FedFNN`, `FedFast` and `FedAvg`. Note that the sampling rate is the fraction of clients which participate in the training at each iteration. We choose 3 different sampling sizes, $\{1\%, 5\%, 20\%\}$. In Figure 4a we present the experiments. For the smallest sample size (Fig. 5a) `FedFast` and `FedAvg` present high level of variability along all the training rounds, while `FedFNN` is stable, converging faster than `FedFast` and `FedAvg` to higher level of accuracy. For the other sample sizes (Fig. 5b and Fig. 5c) the differences in performance between `FedFNN` and the other algorithms still exists but it is increasingly diminished.

6 Discussions and Conclusions

This paper presents `FedFNN`, a new framework in Federated Learning, which speeds up the training of machine learning models, with a focus on the recommender systems. `FedFNN` framework predicts the weight updates for clients not available during training, increasing the performance of the model, especially at the early stage of the process, proving to be not only faster, but with higher accuracy in the same amount of training rounds. Moreover, `FedFNN` is shown to perform optimally in conditions where the sample size is relatively small or when some clients present poor availability. We also demonstrate empirically that training `FedFNN` over input data presenting a high number of clustered preferences (e.g. distinct group of clients with distinguishable preferences) is still beneficial when compared with existing alternatives. Given its novelty, there are several limitations of our model that can be address in the future work. Firstly, it is trained with offline data (ratings or preferences existing before the start of the training), which does not include any online interactions. We introduce `FedFNN`

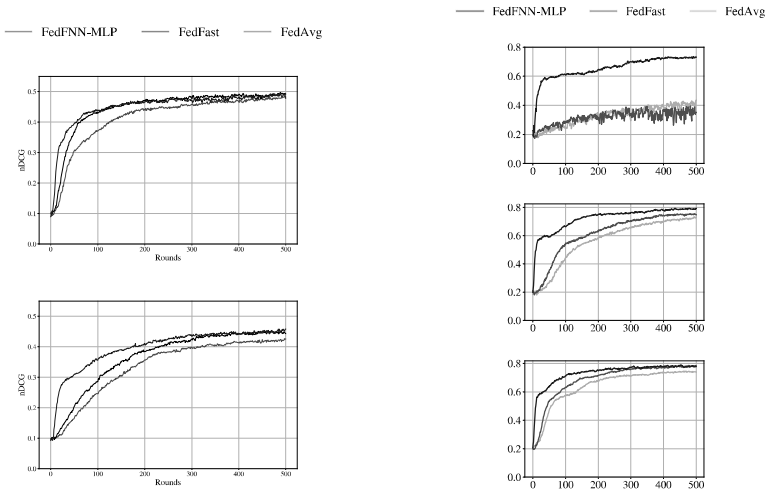


Fig. 4: Simulations for: different configurations of poor availability (left) and 3 distinct sample sizes (right). The models are tested on FedFNN, FedFast and FedAvg.

in the context of recommendation algorithms based on matrix factorisation. Although matrix factorisation remains a popular approach in the literature, a new family of algorithms based on deep neural networks recently gaining traction. In the future, we will consider the adaptation of `FedFNN` to these approaches, such as graph neural networks (GNN) or Deep & Cross Networks (DCN) [22, 21]. `FedFNN` has a significant impact on the first iterations of the model.

Bibliography

[1] Anelli, V.W., Deldjoo, Y., Noia, T.D., Ferrara, A., Narducci, F.: Federank: User controlled feedback with federated recommender systems. In: European Conference on Information Retrieval. pp. 32–47. Springer (2021)

[2] Chen, F., Luo, M., Dong, Z., Li, Z., He, X.: Federated meta-learning with fast convergence and efficient communication. arXiv preprint arXiv:1802.07876 (2018)

[3] Flanagan, A., Oyomno, W., Grigorievskiy, A., Tan, K.E., Khan, S.A., Ammad-Ud-Din, M.: Federated multi-view matrix factorization for personalized recommendations. arXiv preprint arXiv:2004.04256 (2020)

[4] Ha, D., Dai, A., Le, Q.V.: Hypernetworks. arXiv preprint arXiv:1609.09106 (2016)

[5] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016)

[6] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th international conference on world wide web. pp. 173–182 (2017)

[7] Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977 (2019)

[8] Kalloori, S., Klingler, S.: Horizontal cross-silo federated recommender systems. In: Fifteenth ACM Conference on Recommender Systems. pp. 680–684 (2021)

[9] Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S., Stich, S., Suresh, A.T.: Scaffold: Stochastic controlled averaging for federated learning. In: International Conference on Machine Learning. pp. 5132–5143. PMLR (2020)

[10] Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)

[11] Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine **37**(3), 50–60 (2020)

[12] Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks. arXiv preprint arXiv:1812.06127 (2018)

[13] Liang, F., Pan, W., Ming, Z.: Fedrec++: Lossless federated recommendation with explicit feedback. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 4224–4231 (2021)

[14] Lin, G., Liang, F., Pan, W., Ming, Z.: Fedrec: Federated recommendation with explicit feedback. IEEE Intelligent Systems (2020)

[15] Lin, Y., Ren, P., Chen, Z., Ren, Z., Yu, D., Ma, J., Rijke, M.d., Cheng, X.: Meta matrix factorization for federated rating predictions. In: Proceed-

ings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 981–990 (2020)
[16] Lutz, P.: Early stopping-but when. Neural Networks: Tricks of the trade pp. 55–69 (1998)
[17] McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
[18] Muhammad, K., Wang, Q., O'Reilly-Morgan, D., Tragos, E., Smyth, B., Hurley, N., Geraci, J., Lawlor, A.: Fedfast: Going beyond average for faster training of federated recommender systems. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1234–1242. KDD '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3394486.3403176, https://doi.org/10.1145/3394486.3403176
[19] Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. In: Recommender systems handbook, pp. 1–35. Springer (2011)
[20] Wang, J., Liu, Q., Liang, H., Joshi, G., Poor, H.V.: Tackling the objective inconsistency problem in heterogeneous federated optimization. arXiv preprint arXiv:2007.07481 (2020)
[21] Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., Chi, E.: Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In: Proceedings of the Web Conference 2021. pp. 1785–1797 (2021)
[22] Wu, C., Wu, F., Cao, Y., Huang, Y., Xie, X.: Fedgnn: Federated graph neural network for privacy-preserving recommendation. arXiv preprint arXiv:2102.04925 (2021)
[23] Wu, C., Wu, F., Di, T., Huang, Y., Xie, X.: Fedctr: Federated native ad ctr prediction with multi-platform user behavior data. arXiv preprint arXiv:2007.12135 (2020)
[24] Ye, Y., Li, S., Liu, F., Tang, Y., Hu, W.: Edgefed: Optimized federated learning based on edge computing. IEEE Access **8**, 209191–209198 (2020)
[25] Zhang, C., Ren, M., Urtasun, R.: Graph hypernetworks for neural architecture search. arXiv preprint arXiv:1810.05749 (2018)
[26] Zhang, K., Yang, C., Li, X., Sun, L., Yiu, S.M.: Subgraph federated learning with missing neighbor generation. Advances in Neural Information Processing Systems **34** (2021)
[27] Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR) **52**(1), 1–38 (2019)