Deep Federated Recommendation System

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Artificial Intelligence
Master of Science in Computer Engineering

By

Archil Beridze
Omkar Suryakant Naik
Utsav Chandresh Savaliya
Charanveer Singh

May, 2023

APPROVED

DocuSigned by:

Mahima Agumbe Suresh

E36A1695C9BC46E...

5/12/2023

Mahima Agubme Suresh, Project Advisor

Abstract**—With the rapid development of deep learning algorithms in recent years, intelligent systems now play an intrinsic role in most of today's industries. Consequently, there is high demand in deep-learning based applications which seek patterns in the user data to maximize the user experience and minimize the costs of a company. One such application is recommendation systems, which allow companies to market specific products directly to individuals that have the highest probability of being interested in the said product. In recent years we have seen a shift from primitive recommendation algorithms, such as collaborative filtering and matrix factorization to more complex deep learning approaches, which are capable of more effectively processing large volumes of data, while also exhibiting better performance. Such recommendation systems are employed by industry giants from virtually every field, such as digital marketing, streaming, video game industry and many more. However, with the increase in volumes of data, the concern for privacy also increases, and recently we have seen many cases where companies fail to protect the data of their consumers. In this report we propose a deep learning-based recommendation system that provides recommendations to a user based on their previous activity. To account for the increasing concern of data privacy, we will be employing a federated learning approach, using which we prevent users from directly sending their data to the server, avoiding any risk of it being intercepted by third parties.**

Keywords: recommendation systems, deep learning, federated learning, collaborative filtering

I. INTRODUCTION

A. Project Motivation

Machine learning has drawn a lot of attention as it is a very effective business strategy, demonstrated by certain influential tech giant companies and their success with it. The recommender systems (RS) are arguably one of the more popular applications of machine learning, used by Google, Facebook, Amazon, and Netflix to expand their businesses. Although traditional recommendation approaches, such as collaborative or content-based filtering have shown excellent results for RS, nowadays the vast majority of companies utilize neural network-based approaches capable of analyzing larger volumes of data and thus producing more accurate recommendations. Neural collaborative filtering is arguably one of the most popular approaches that are driven by deep learning [1]. Apart from their increased efficiency, neural-based RS has another great advantage over their classical counterparts - they allow the utilization of federated learning algorithms to ensure data privacy [2]. As of now, the existing federated learning approaches allow the aggregation of just deep learning models.

B. Problem Statement

Data is often regarded as one of the most valuable resources of the 21st century. In contrast with other valuable resources, the value of data is not in its scarcity, but rather in its utility. As the methods and algorithms for analyzing the data increase, so does the utility and thus the value of the data. However, with increased value comes an increased demand for privacy. In recent years there have been many cases where even the large corporations are found to be not handling or storing sensitive client data properly, often leading to it leaking to third parties. To mitigate this, researchers and engineers propose various techniques to increase data privacy, with federated learning being one such approach.

C. Current state-of-the-art

Federated learning allows companies to train their machine learning models on user data without having to handle it directly. Instead of uploading the data to a server, the user downloads the machine learning model, trains it on their data locally, and then uploads the model back to the server, where together with other models it gets aggregated through a federated averaging algorithm. This federated learning technique was proposed by Mcmahan et. al. [2] and remains to be the state-of-the-art approach to this day.

Neural collaborative filtering and its derivative model architectures are regarded as current state-of-the-art approach for RS. For instance, a similar architecture is utilized by Facebook for their DLRM (deep learning recommendation system), which utilizes the combination of multi-layer perceptron (MLP) and matrix factorization provided as user/item embedding layers.

D. Proposed approach

With the rise in demand for recommendation systems in many industries such as streaming, online marketing, digital entertainment, and many others (where user data could be quite sensitive), we believe that recommendation systems can greatly benefit from approaches that focus on data privacy, such as federated learning. Although there already are quite a few publications involving federated recommendation systems, the area of research is quite new and there are not many open-source projects available for experimentation. In our project we demonstrate a recommendation system that can be trained using federated training process.

II. NEURAL COLLABORATIVE FILTERING

A. Model Architecture

Neural collaborative filtering (NCF) is a recommendation system architecture proposed by He et. al. [1]. The model fuses multi-layer perceptron with a generalized matrix factorization model as it can be seen in Figure 1. Users and items are represented in terms of embedding vectors, for GMF layer they are combined through dot product, and for MLP layer they are combined through concatenation. The outputs of GMF and MLP layers are then concatenated and fed through the final, output layer (or layers depending on model configuration). In their paper researchers use MLP of depth 4 as their output layer.

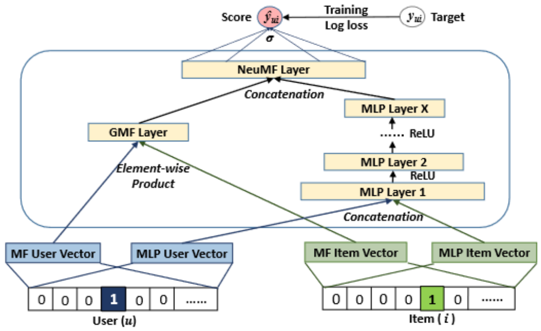

Figure 1: NCF model architecture

Since NCF is used to model user-item interactions, the target values in the training data are given as vector containing binary values, therefore binary cross-entropy is used as the loss function.

B. Dataset

The authors of the paper use Movielens and Pinterest datasets to train and evaluate their model. Since our implementation uses NCF model, we will also utilize the two aforementioned datasets to benchmark our implementation. The characteristics of the datasets are provided the Table 1.

Dataset	# Interactions	# Users	# Items
Movielens	1,000,209	6,040	3,706
Pinterest	1,500,809	55,187	9,916

C. Evaluation metrics

In their paper the authors use hit ratio and normalized discounted cumulative gain as the metrics for evaluating their model.

Hit ratio (HR) is a metric that indicates if a test item is presented in a list of top k recommendations for a given user. Larger values of hit ratio indicate better performance of a recommendation system.

Normalized discounted cumulative gain (NDCG) is a metric that indicates the relevance of the recommendation generated by the system. It is calculated using formula:

$$NDCG_k = \frac{1}{N}\sum_{i}^{k} \frac{\ln(2)}{\ln(i+2)}$$

Where *k* is number of top recommendations we use to check relevance for, *i* indicates the rank of the item in top-k list, *N* is the total number of test items.

For evaluating the federated model, we will be using the same two metrics and comparing our results with the authors`.

III. FEDERATED LEARNING

A. Overview

As mentioned in the introduction, federated learning is first and foremost a privacy feature, which prevents malicious users from obtaining sensitive client data even if the compromise the application server. Privacy is ensured by the fact that users never directly send their personal data to the server. Instead, the modeling on user data is done by following procedure:

1. user downloads the deep learning model on their device
2. the model is trained locally on device
3. the model is sent to server through secure, encrypted channel
4. server aggregates the states of models across different users

At no point during this process does user send their data to the server or share it with any other clients in the system. The aggregation of models' states can be done through different algorithms, with the most popular and straightforward one being federated averaging (FedAvg), which also happens to be the method we will be employing in our project.

B. Related Works

Federated learning was first proposed by Google researchers McMahan et. al [2]. Google has also developed a framework for training and deploying federated deep learning models as a part of their Tensorflow framework, called Tensorflow Federated (TFF).

Due to the rise in concerns for the data privacy, there has been quite extensive research done in the field of federated learning in recent years, including in the context of federated recommendation systems [3][4][6]. For instance, Liang et. al. propose federated RS which trains the model by averaging the user data, given in terms of product ratings, and then randomly sampling from it during training [4]. Others, like Dogra et. al., propose a more sophisticated system, capable of providing state-of-the-art performance at a decreased memory usage [3]. Other researchers raise a question about cyber-attacks which could specifically target federated systems. Duan et. al. demonstrate how a malicious user could poison the federated learning system by injecting it with falsified user data [5]. Chen et. al. propose a federated learning approach that provides better recommendations based on the geolocation of the user [6]. They achieve this by clustering the users based on their product preference and training a model by also considering the geographical location of the user.

C. Pitfalls

While federated learning is an impressive approach for ensuring data privacy, it introduces some additional concerns when it comes to training the models. One possible concern, as was mentioned in the previous section, is malicious users injecting falsified data into the system. But there are also other challenges with the training process, not related to malicious users deliberately hindering the system's performance.

One such challenge is increasing the accuracy of federated models. Federated neural networks are trained on data that is distributed across different users. Since the distribution is not guaranteed to be uniform, it is difficult for the model to fully capture the data representation [12]. This is especially prevalent if the training is done on a small subset of randomly sampled users. To tackle this problem, we use a sampling strategy that will be discussed in the following chapter.

IV. Project Design

A. Project structure

Our implementation consists of two parts – first a module, which contains the implementation of NCF algorithm, function to evaluate the model and train it through federated learning. Second part acts as a driver of the module, it simulates the interaction between server and a client.

```
project/
├── src/
│   └── federeco/
│           ├── models.py
│           ├── client.py
│           ├── train.py
│           ├── eval.py
│           └── config.py
│       ├── main.py
│       ├── server.py
│       ├── server.py
│       └── dataset.py
```

federeco module, as stated before, acts as the core part of the federated recommendation system. It consists of following files:

1. models.py – contains implementation of NCF.
2. client.py – contains abstract *Client* class. In our implementation, training data of NCF model must be wrapped by *Client* class.
3. train.py – contains functions that train the NCF model via federated training. FedAvg algorithm can also be found here.
4. eval.py – contains functions to calculate HR and NDCG
5. config.py – contains model parameters

B. Server and Client

Other files outside of *federeco* module simulate the interaction between user and the server. Main driver of the simulator is *sever.py* file which performs following actions:

1. Loads training datasets
2. Initializes *Client* objects and sets them up with training data
3. Launches federated training process
4. Evaluates the trained model
5. Generates recommendations for users

Client class defines the user of the application. Clients must be set-up with a unique identifier, initialized with their respective item vector and must implement *train* method that accepts server model as a parameter and trains it locally on client's item vector. *federeco* module defines abstract *Client* class that serves as a template.

C. Training Process

```
Algorithm 1: TrainingProcess
$shuffle(allclients)$;
for $i = 1$ to $E$ do
    $clients \leftarrow sample(allclients,\ n)$;
    for $j = 1$ to $numclients$ do
        $weights_j \leftarrow train(clients_j,\ e)$;
    end
    $serverweights \leftarrow FedAvg(weights)$;
end
return $serverweights$;
```

Federated training process consists of several steps:

1. Sample *n* clients from dataset
2. Train sampled clients on their own dataset for *e* epochs
3. Collect the weights of the models trained locally by clients
4. Aggregate weights through FedAvg
5. Repeat *E* times

D. Client sampling

In practice, due to limited bandwidth, federated learning systems often initiate training process on a small subset of total clients. This part is simulated in part 1 from the list given above. However, during implementation it was discovered that random sampling was not the best strategy for training a federated model. The main issue is that when sampling from the list of clients with length *L* and subset size of *S*, if the values of *S*, *L* are too small, not all clients will be sampled from the list by the end of training process. This problem is known as *coupon problem* in probability theory.

During the fine-tuning of the model, it was noticed that when sampling the list of 6040 users with sample size of 50, approximately 25% of clients were not being sampled when training over 500 epochs. One way to ensure that all clients will be sampled (with some confidence *c*) is to increase the sample size and number of training epochs. However, the sample size is constrained by the bandwidth of the system, while increasing the number of epochs can lead to model being overfit and skewed towards samples that it sees most often.

To guarantee that the sampling of users is uniform, instead of randomly sampling, we create a queue from the total list of users. During each round of federated training, we take *n* number of users from the start of the queue, initiate training process for these *n* users, and append them at the end of the queue. This operation is equivalent to rotating the list left with shift size of *n*. The number of times model sees the data from a single client can be changed by tuning parameters *e* and *E*.

V. Implementation Details

We construct NCF model depicted in Figure 1 with following parameters: for MF layer we use latent dimension of 16, for MLP we use hidden layers with following number of nodes: {64, 32, 16, 8}.
Federated NCF also introduces two additional parameters:

1. *n* - indicates number of sampled users during a single training round.
2. *e* - number of epochs for local training of sampled clients

Due to bandwidth limitations in federated learning, it is not feasible to train the server model on all clients in the system at once, therefore a subset of clients is chosen for single training round. During each global training epoch, we pick *n* number of clients to train locally for *e* epochs. Optimal values for *n* and *e* are provided in next section.

During data preprocessing, we sample the data in such way that we have 4 negative samples per each positive sample for every unique client ID in the dataset. We train federated NCF for 400 epochs with batch size of 64. AdamW [13] was used as an optimizer with learning rate of 0.001. Binary cross-entropy function was used to calculate loss.

The model is implemented in PyTorch framework, but Tensorflow implementation is also available on project's GitHub repository under *tensorflow* branch. Training and evaluation were done on NVIDIA GeForce GTX 1060 GPU.

VI. Evaluation methodology

For evaluating FedNCF model we use same metrics as He et. al. use for validation, namely hit rate and normalized discounted cumulative gain [1].

Both metrics are measured with respect to top 10 recommendations. Model evaluation was done across different number of epochs, as well as for different values of *n* and *e*.

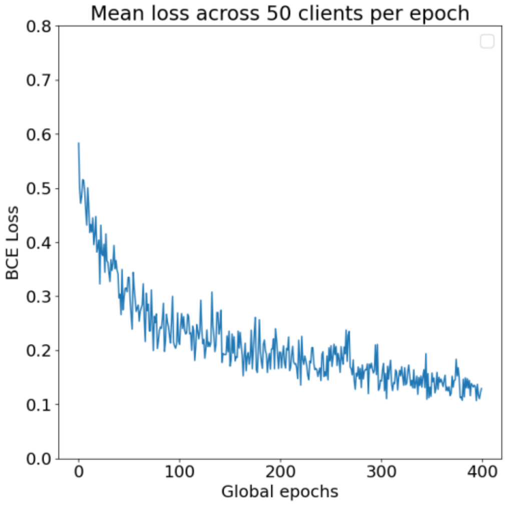

Figure 2: FedNCF training loss

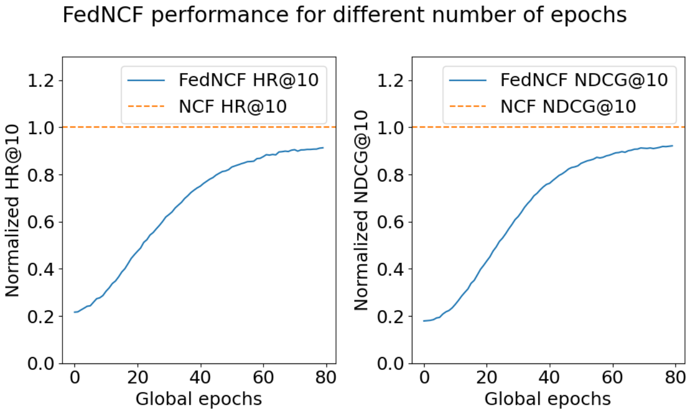

Figure 3: FedNCF performance across different epochs

Figure 2 depicts mean binary cross entropy loss across the clients sampled during each global training epoch. As illustrated, the training loss steadily decreases during first 400 epochs, however, training model further resulted in diminishing results, therefore we set number of training epochs to 400 during our further evaluations.

In their paper, He et. al. mention that after certain number of training epochs, they noticed that although the loss kept decreasing, the model's recommendation quality worsened. To verify this is not the case with our training process, we also measure the model's accuracy on test data across different epochs. As depicted on Figure 3, model's performance steadily increases for first few hundred epochs, with the curve flattens at around 400 epochs, confirming that the model does not in fact overfit and that 400 epochs is a valid number for global training epochs.

We also test model's performance with respect to the two new parameters introduced by federated training process: *n* and *e*. Figure 3 depicts the change in HR and NCDG for different values of *n* and *e*.

As we can see in Figure 4, parameter *n* has negligible effect on model's accuracy. It does however affect the time and bandwidth requirements of the training process. Smaller values of *n* translate to larger number of iterations needed for convergence, while larger values of *n* require more bandwidth. The fact that model's performance is not dependent on parameter *n* is quite expected, since it does not change the number of times the data of each client gets fed through the model. Sample size only affects number of times weight aggregation happens on the server side, which seems to have no effect on the recommendation quality of the model.

However, parameter *e* does affect model's performance, this is especially evident for smaller numbers of global epochs. For 400 epochs however, it seems that 3 local training epochs is the most optimal value. Larger values of *e* are likely to cause overfitting, while also increasing the training time.

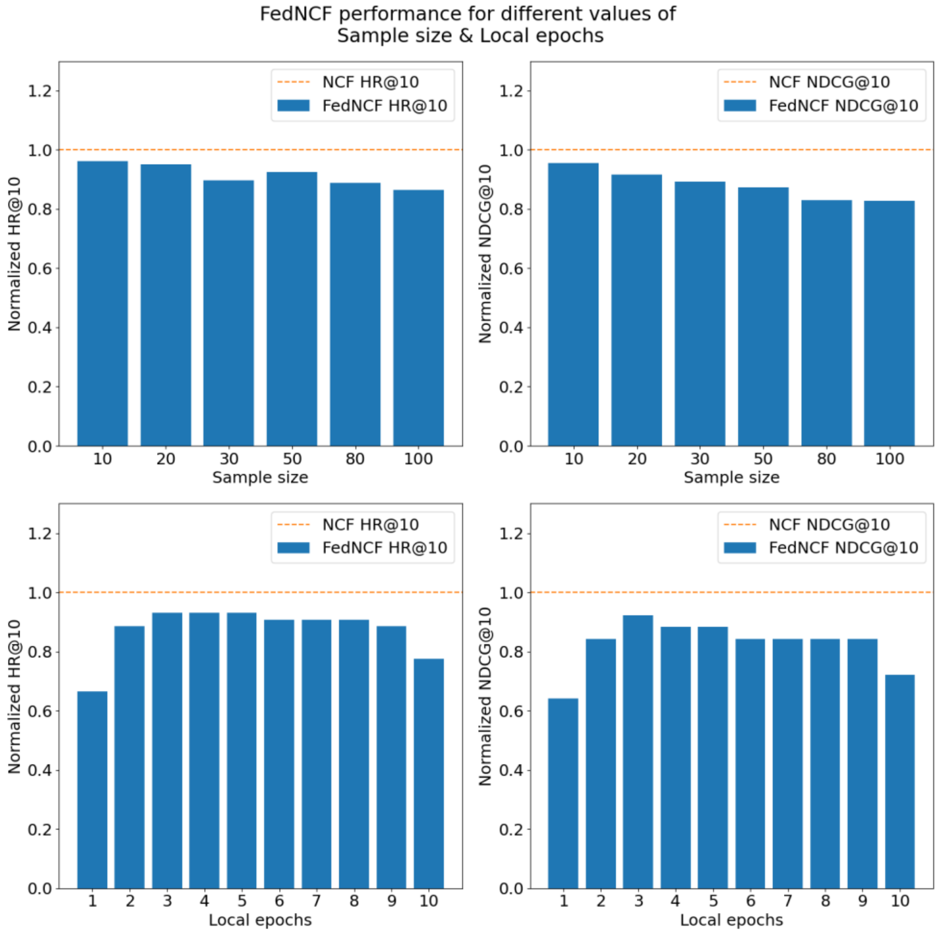

Figure 4: FedNCF performance w.r.t. different values of n and e

VII. Conclusion

In this report we have demonstrated that NCF-based recommendation system can be trained through federated averaging technique with minimal loss in model's performance. We have also shown what effect the newly introduced hyperparameters have on the performance of the model. We hope that our project can serve as a solid baseline with anyone who wishes to experiment with federated recommendation systems or federated learning in general.

One drawback that FedNCF has is that it's a static model, once trained, it becomes impossible to generate recommendations for new users or add any new items to the system [14]. This is true for any recommendation system that utilizes matrix factorization, including original NCF implementation. However, there are some existing techniques that allow us to online-update the parameters of the MF layer and introduce new users/items to the system [14][15]. Same concepts can also be applied to federated NCF as well.

REFERENCES

1. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural Collaborative Filtering,” Proceedings of the 26th International Conference on World Wide Web - WWW ’17, 2017, doi: 10.1145/3038912.3052569.
2. H. Mcmahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera, “Communication- Efficient Learning of Deep Networks from Decentralized Data.”
3. M. Dogra, B. P. Meher, P. V. Mani and H. -K. Min, "Memory Efficient Federated Recommendation Model," *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, 2022, pp. 139-142, doi: 10.1109/ICSC52841.2022.00028.
4. G. Lin, F. Liang, W. Pan and Z. Ming, "FedRec: Federated Recommendation With Explicit Feedback," in *IEEE Intelligent Systems*, vol. 36, no. 5, pp. 21-30, 1 Sept.-Oct. 2021, doi: 10.1109/MIS.2020.3017205.
5. D. Rong, S. Ye, R. Zhao, H. N. Yuen, J. Chen and Q. He, "FedRecAttack: Model Poisoning Attack to Federated Recommendation," *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2643-2655, doi: 10.1109/ICDE53745.2022.00243.
6. S. Duan, D. Zhang, Y. Wang, L. Li and Y. Zhang, "JointRec: A Deep-Learning-Based Joint Cloud Video Recommendation Framework for Mobile IoT," in *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1655-1666, March 2020, doi: 10.1109/JIOT.2019.2944889.
7. W. Li, H. Chen, R. Zhao and C. Hu, "A Federated Recommendation System Based on Local Differential Privacy Clustering," 2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation, 2021, pp. 364-369
8. Q. Chen and J. Qin, "Research and implementation of movie recommendation system based on deep learning," 2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), 2021, pp. 225-228, doi: 10.1109/CEI52496.2021.9574461.
9. K. Järvelin and J. Kekäläinen, “Cumulated gain-based evaluation of IR techniques,” ACM Transactions on Information Systems, vol. 20, no. 4, pp. 422–446, Oct. 2002, doi: 10.1145/582415.582418.
10. K. Chandiramani, D. Garg, and N. Maheswari, “Performance Analysis of Distributed and Federated Learning Models on Private Data,” Procedia Computer Science, vol. 165, pp. 349– 355, 2019, doi: 10.1016/j.procs.2020.01.039.
11. G. Su, "Federated Online Learning Based Recommendation Systems for Mobile Social Applications," 2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), 2021, pp. 936-939
12. W. Huang, T. Li, D. Wang, S. Du, and J. Zhang, “Fairness and Accuracy in Federated Learning,” Dec. 18, 2020.
13. I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” arxiv.org, Nov. 2017
14. S. Rendle and L. Schmidt-Thieme, “Online-updating regularized kernel matrix factorization models for large-scale recommender systems,” Proceedings of the 2008 ACM conference on Recommender systems, Oct. 2008, doi: https://doi.org/10.1145/1454008.1454047.
15. Christopher C. Johnson Muqeet Ali and Alex K. Tang. Parallel collaborative filtering for streaming data. http://www.cs.utexas.edu/ cjohnson/, December 2011

Acknowledgements

The authors are deeply indebted to Professor Mahima Agumbe Suresh for her invaluable comments and assistance in the preparation of this study.