Deep Federated Recommender Systems

A Project Report Submitted
in Partial Fulfillment of Requirements
for the Degree of

Bachelor of Technology with Honors

by
Rishabh Jain(2019CSB1286)

Department of Computer Science & Engineering
Indian Institute of Technology Ropar
Rupnagar 140001, India
May 2023

Abstract

With data being called the 'oil' of the 21st century, it is becoming an invaluable commodity. Data is primarily being used on a large scale by industries worldwide to train extensive artificially intelligent systems. This data, however, often contains user sensitive information which causes privacy concerns. Furthemore, it is becoming increasingly expensive to train large scale models due to ever increasing number of users. Federated Learning is a distributed machine learning paradigm which addresses the issue of data privacy, as well as offsets the cost of compute requirements to edge devices. Federated learning algorithms help train machine learning models without sharing the user data with the service providers. In this work, we focus on the application of user privacy in federated recommender systems.

Acknowledgements

And I would like to acknowledge Dr. Shweta Jain, honors project supervisor for her guidance and support for the entirety of this project. Without her invaluable encouragement and engagement none of this would've been possible.

Honor Code

I certify that I have properly cited any material taken from other sources and have obtained permission for any copyrighted material included in this report. I take full responsibility for any code submitted as part of this project and the contents of this report.

Rishabh Jain (2019csb1286)

Certificate

It is certified that the B. Tech. Honors project "Client Selection in Deep Federated Recommender Systems" has been done by Rishabh Jain(2019csb1286) under my supervision. This report has been submitted towards partial fulfillment of B. Tech. Honors project requirements.

Dr. Shweta Jain
Project Supervisor
Department of Computer Science & Engineering
Indian Institute of Technology Ropar
Rupnagar-140001

Contents

Contents v

List of Figures vii

1 Introduction 1

1.1 The Compute Problem 1

1.2 The User Privacy Problem 1

1.3 Application in Recommender Systems 2

2 Recommender Systems 3

2.1 Introduction 3

2.2 Types of recommender system types 3

2.2.1 Content-based Recommender Systems 3

2.2.2 Collaborative Recommender Systems 4

2.2.3 Hybrid Recommender Systems 4

2.3 Collaborative Filtering 4

2.3.1 Low-Rank Matrix Factorization 4

3 Federated Learning 6

3.1 Introduction 6

3.2 FedRec: Federated Recommender System with Explicit Feedback 6

3.2.1 Storing user data and user embeddings 7

3.2.2 Loss calculation and gradient flow 7

3.2.3 Client Privacy Preservation 8

3.2.4 FedRec algorithm 8

3.3 Experiments . . . 9
3.4 Results . . . 9

4 Client Selection in Federated Learning 10
4.1 Bottleneck in Federated Learning . . . 10
4.2 Client Aware Selection Strategies . . . 10
4.2.1 Sub-modular maximization . . . 10
4.2.2 Max-loss based . . . 11
4.2.3 Clustering . . . 11
4.3 Experiments . . . 11
4.4 Results . . . 12

5 Deep Federated Recommender Systems 13
5.1 Introducing Deep FedRec . . . 13
5.1.1 Client Model . . . 13
5.1.2 Neural Network Architecture . . . 14
5.1.3 Experiments . . . 14
5.1.4 Results . . . 15
5.2 Prior Work: Deep Autoencoder based Collaborative Filtering . . . 16
5.2.1 Model Architecture . . . 16
5.2.2 Loss Function . . . 16
5.2.3 Dense Refeeding . . . 17
5.3 Introducing Deep Federated AutoEncoder Recommender System . 18
5.3.1 User Model . . . 18
5.3.2 Gradient flow architecture . . . 18
5.3.3 Loss Function . . . 19
5.3.4 Experiments . . . 19
5.3.5 Results . . . 20

6 Conclusions 21

References 22

List of Figures

3.1 FedRec Architecture 7
3.2 FedRec on ML100K dataset 9

4.1 Submodular function G 11
4.2 Subset generation algorithm by maximizing G 11
4.3 FedRec accuracy vs epochs for various selection strategies 12

5.1 Deep FedRec Architecture 14
5.2 FedRec Client Model 15
5.3 Deep FedRec: Accuracy vs Epochs 15
5.4 Deep Autoencoder Architecture. Ref- Kuchaiev and Ginsburg [2017] 17
5.5 Federated Deep Autoencoder Architecture 19
5.6 Federated Deep Autoencoder Architecture Accuracy vs Epochs . . 20

Chapter 1

Introduction

Federated Learning is a distributed machine learning paradigm. It solves the following two problems:

1.1 The Compute Problem

With the number of edge devices growing exponentially, it is becoming harder to train large scale AI based services centrally on the server. Industry is therefore focussed on offsetting some of this compute cost on the edge devices. The federated learning thus helps in the development of such services at a reduced cost.

1.2 The User Privacy Problem

Protection of user privacy is of utmost concern in the world today. Data used to train extensive AI based services is collected from the users, which often contains sensitive user information. With the techniques in federated learning, one can train AI models without requiring to extract user data from the edge devices.

1.3 Application in Recommender Systems

Federated Learning is a boon to recommender systems, which are an integral part of almost any service today. It helps companies to train complex recommender systems at a reduce cost, and while following user privacy restrictions set by the Government.

Chapter 2

Recommender Systems

2.1 Introduction

Almost every user centric client services such as Netflix, Amazon, YouTube, etc use algorithms to suggest content based on the user's interest. These algorithms are a part of recommender systems. These systems use user's prior watch history, item ratings, and information from adverts to predict content which the user might like.

The first stage of this work was to explore recommender system algorithms extensively and then use the gained knowledge for application in federated setting.

2.2 Types of recommender system types

Three primary types of approaches to recommender systems are described below:

2.2.1 Content-based Recommender Systems

Content based recommender systems learn item features to recommend them. It creates user profile based on the items user has seen before. The user profile is then used to predict items that the user might like in the future.

2.2.2 Collaborative Recommender Systems

These recommender systems attempt to learn item features based on user interests. It is based on the hypothesis that users with similar tastes in the past will have similar tastes in the future. Thus, item features are automatically learnt from correlating user interests, and does not use information of the item separately for training.

2.2.3 Hybrid Recommender Systems

As the name suggests, hybrid recommender systems utilize the strengths of content-based and collaborative recommender systems. It uses item features and user profiles to predict content. They give much more accurate results than content-based or collaborative recommender systems.

2.3 Collaborative Filtering

In our work, we primarily focus on collaborative filtering for item recommendation Zhang et al. [2014] . In this section, we discuss collaborative filtering algorithm.

Given a set of users U, a set of items V and a sparse matrix of ratings R, our task is to predict the missing entries in R. The matrices U and V need to be learnt to identify user and item features.

2.3.1 Low-Rank Matrix Factorization

A well known algorithm for collaborative filtering is the low-rank matrix factorization. Let k be the latent feature dimension of item and user vectors. U is a matrix of dimension $N \times k$, where N is the number of users. V is a matrix of dimension $M \times k$, where M is the number of items. By definition,

$$R = UV^T \tag{2.1}$$

where R is the rating matrix of dimension $N \times M$ available from the dataset. Our goal is to learn the matrices U and V, in order to learn user and item features. Their product will then help predict the the missing user-rating pairs in R.

This can be modelled as a low-rank matrix factorization problem. However, splitting large matrices can be computationally expensive using algebraic methods such as singular value decomposition. We can instead model this problem as a least squares optimization problem, where the loss

$$L = \min_{U \in \mathbb{R}^{N \times k}, V \in \mathbb{R}^{M \times k}} ||R - UV^T||^2 \tag{2.2}$$

can be minimized using the algorithms such as gradient descent. Regularization terms can be added to prevent over-fitting.

Chapter 3

Federated Learning

3.1 Introduction

Federated learning is a distributed machine learning paradigm. It commonly of a network of edge devices and an server. It eliminates the need to transfer sensitive user data to the server, thus reducing compute and network costs; and preserving user privacy.

3.2 FedRec: Federated Recommender System with Explicit Feedback

Lin et al. [2021] describe an architecture to implement collaborative filtering in a federated setting.

In the previous chapter, we describe collaborative filtering as a low-rank matrix factorization problem.

Given a set of users U, a set of items V and a sparse matrix of ratings R, our task is to predict the missing entries in R. The matrices U and V need to be learnt to identify user and item features.

3.2.1 Storing user data and user embeddings

Now, to prevent user privacy leakage, the user data is stored on the user, along with the user embeddings U_i. The item embedding matrix V is stored on the server and is shared with the clients periodically.

3.2.2 Loss calculation and gradient flow

Algorithm 1 describes the overall flow of gradients through the network. Here is a brief flow:

1. First, the user performs local training steps using the rating vector R_i, user embedding U_i and item embedding matrix V.
2. Loss is computed. Gradients ∇U_i, ∇V are calculated.
3. ∇U_i is used to updated the user embedding. ∇V is sent to the server for aggregation.

Refer Figure 3.1.

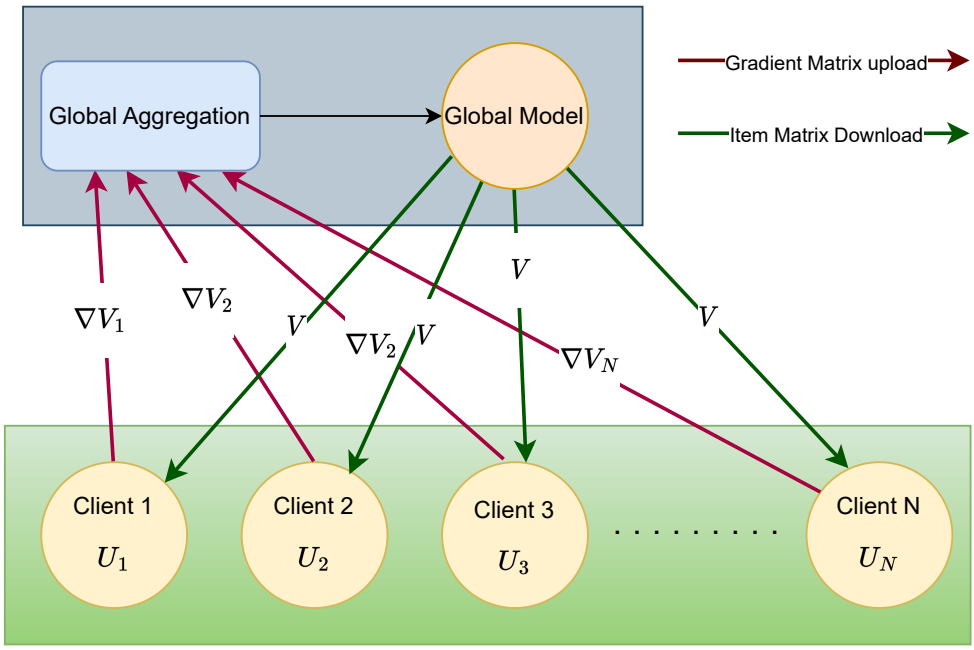

Figure 3.1: FedRec Architecture

3.2.3 Client Privacy Preservation

We can see that only item vector gradients, and no data or user embedding is sent to the server. This preserves client privacy, as well as saves compute and network resources to transfer large amounts of data to the server.

However, if a certain client has not rated certain movies, then the gradient for those items will be 0. This might reveal certain user specific information. For this, we randomly select a subset of items and fill them with some values representative of user's interests.

1. Averaging filling: Select a subset of non rated items, and assign them a rating equal to that of average of rated items.

2. Hybrid filling: Use the user embedding U_i to predict the missing ratings and then perform training to obtain gradients.

3.2.4 FedRec algorithm

We summarize the algorithm described by Lin et al. [2021] in Algorithm 1

Algorithm 1 FedRec Algorithm with Explicit Feedback on Server

Initialize U_i for each client $i = 1, 2, 3...n$ on the client
Initialize V_j for each item $j = 1, 2, 3...m$ on server
Initialize n (number of planning steps)
loop for each t = 1, 2, 3.... T:
    loop for each client i:
        Download feature matrix V from server
        $\nabla V_{i,\forall j} \leftarrow TrainClient(U_i, V, R_i)$, where R_i is local user rating data
        Share gradient $\nabla V_{i,j}$ for each item j with server
    end loop
    loop for each item j:
        $\nabla V_j \leftarrow \underset{i}{WeightedAverage}(\nabla V_{i,j})$
        Gradient update: $V_j \leftarrow V_j - \gamma \nabla V_j$
    end loop
end loop

3.3 Experiments

FedRec was implemented on ML100k dataset containing 1000 users and 1700 movies. The algorithm was run for 400 epochs.

3.4 Results

FedRec gave an accuracy of 59.7% in 400 epochs on the dataset.

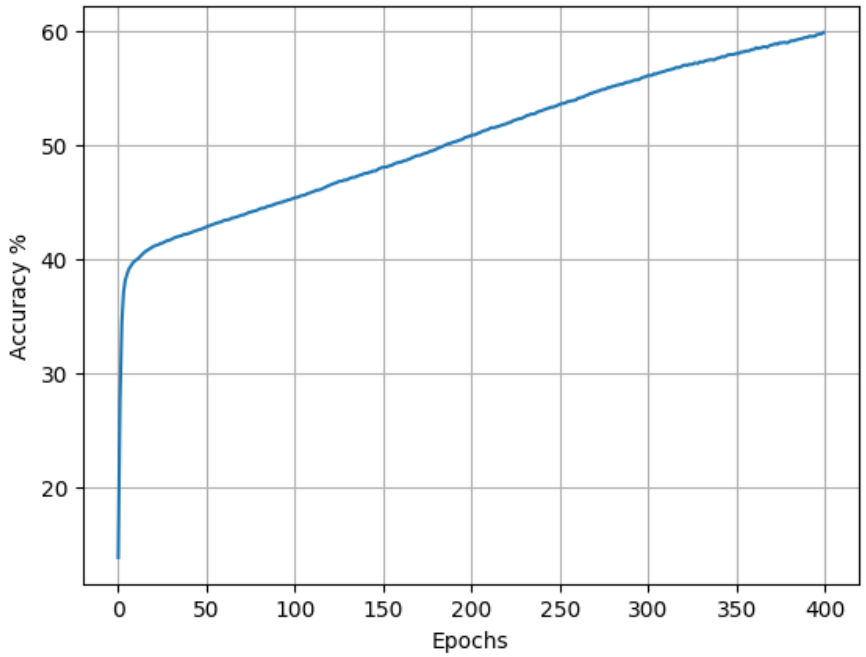

Figure 3.2: FedRec on ML100K dataset

Chapter 4

Client Selection in Federated Learning

4.1 Bottleneck in Federated Learning

While a large portion of the computation requirements are offset to edge devices, the cost of communication remains a significant bottleneck in federated learning. Often, large weight matrices or gradients need to be transferred over the network. Several devices are often disconnected with the server, or have poor network connectivity.

We thus explore client selection strategies, which involve selecting a subset of client based on selection strategies. Our aim is to reduce communication costs, while minimizing performance loss.

4.2 Client Aware Selection Strategies

We explore three strategies of client selection, which are as described in the following subsections.

4.2.1 Sub-modular maximization

Balakrishnan et al. [2022] develop a techique to select a subset of clients which best represent the entire client set using their communicated gradients.

$$\sum_{k \in [N]} \min_{i \in S} \left\| \nabla F_k(v^k) - \nabla F_i(v^i) \right\| \triangleq G(S).$$

Figure 4.1: Submodular function G

$$S \leftarrow S \cup k^*, \;\; k^* \in \underset{k \in \text{rand}(V \setminus S,\ \text{size}=s)}{\text{argmax}} \left[\bar{G}(S) - \bar{G}(\{k\} \cup S) \right]$$

Figure 4.2: Subset generation algorithm by maximizing G

Figure 4.1 and 4.2, which describe the submodular function and the subset generation algorithm, are taken from Balakrishnan et al. [2022]

4.2.2 Max-loss based

Jee Cho et al. [2022] suggest a client selection method to select a subset of top-k clients with maximum loss. We implement this strategy by first selecting a random subset of clients from the pool, and then choosing the top-k clients to improve efficiency. Such an approach does not cause much variation in performance, which was verified experimentally.

4.2.3 Clustering

In this approach, we cluster the clients using k-means based on their gradients received, and pick the centers of the as the set of clients to be used in the training set.

4.3 Experiments

We implemented the above three client selection strategies on FedRec based collaborative filtering. The models were trained on the ML100k MovieLens dataset. We set subset size as 25, and collect all gradients from clients at intervals of 20 epochs. The training was run for a total of 400 epochs.

4.4 Results

In figure 4.3, we can observe that choosing a subset of clients did not cause a significant drop in performance of the model, while reducing the network cost of training. The three selection strategies performed quite similar, with the clustering approach outperforming max loss and submodular maximization.

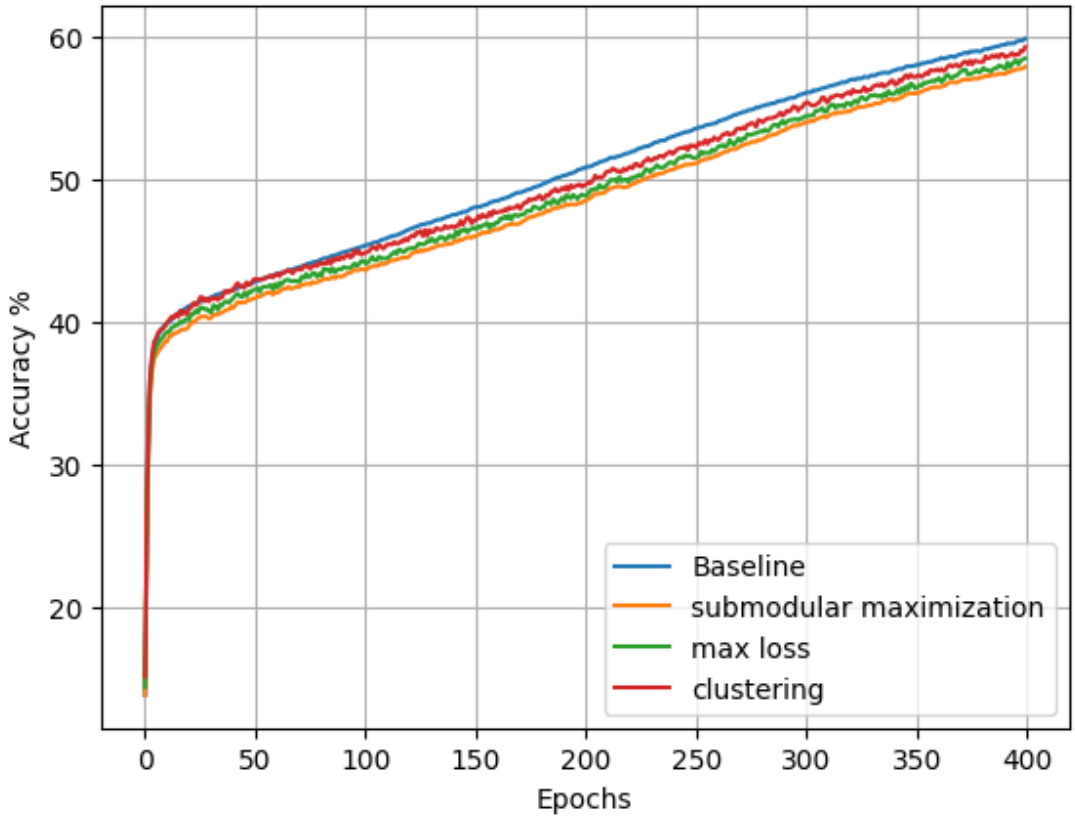

Figure 4.3: FedRec accuracy vs epochs for various selection strategies

Chapter 5

Deep Federated Recommender Systems

Deep Neural Networks are machine learning techniques to learn complex distributions in data. With the assumption that the data is identically and independently distributed(IID), deep neural networks find extensive applications. In this chapter, we aim to apply deep neural network in a federated setting. Specifically, we design two architectures for deep federated recommender systems.

5.1 Introducing Deep FedRec

Till now, we were considering linear mappings between user embedding U and item embeddings V through dot product to predict ratings. Now, we aim to apply non-linearity in the process to increase the capacity of our model.

However, the a significant challenge while training deep neural networks in a federated setting is that distributed data often loses its IID property.

5.1.1 Client Model

As before, the item matrix V will be stored on the server and will be communicated to the client at regular intervals. The client will now maintain a deep neural network locally which contains user embedding U_i in its weights.

The model takes as input both the item and user embedding to predict a rating vector R_i^p. Loss will be computed against the locally stored R_i ground truth vector, and the gradients for the neural network ∇U_i and ∇V_i will be computed. ∇U_i is used to update the user model, whereas ∇V_i is sent to server for aggregation. Refer figure 5.1.

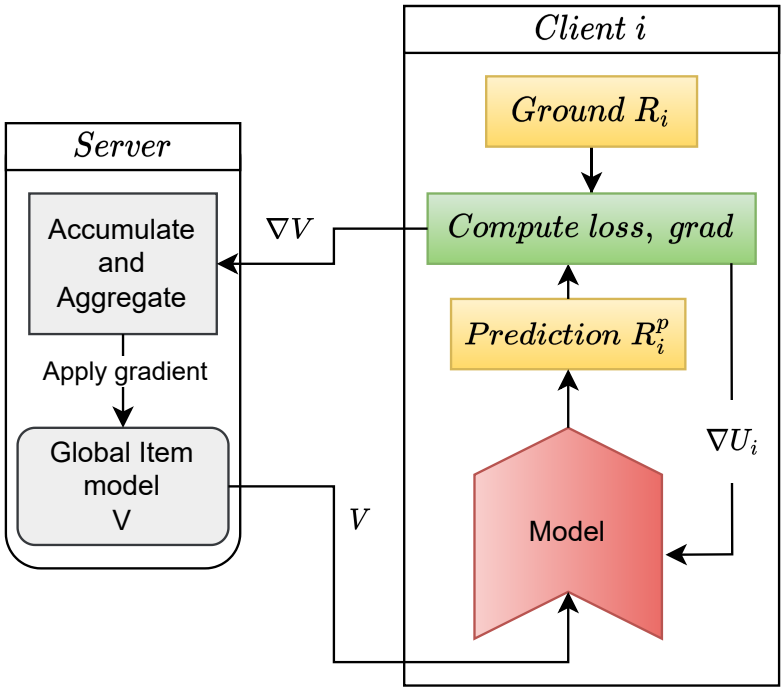

Figure 5.1: Deep FedRec Architecture

5.1.2 Neural Network Architecture

Refer Fig. 5.2 for the model architecture. The model takes as input the item vector for the model, and outputs the predicted rating for the item.

5.1.3 Experiments

The model was implemented using PyTorch and Python.

Dataset The model was trained on the ML100K dataset from MovieLens. The dataset contains ratings from 1000 users on 1700 movies.

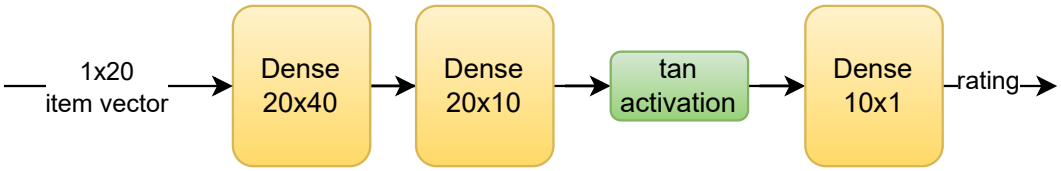

Figure 5.2: FedRec Client Model

Test-Train Split 30% of the rated items were masked from each user's rating vector to create the training vector. The original vector was used to calculate the test accuracy at the end of each epoch.

User learning rate	0.01
Global learning rate lambda	0.01
Epochs	300
Optimizer	Adam

Table 5.1: Training parameters for Deep FedRec

5.1.4 Results

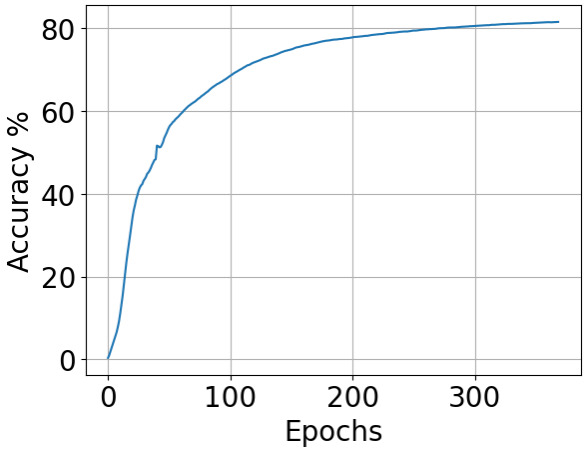

Figure 5.3: Deep FedRec: Accuracy vs Epochs

We can see from Figure 5.3, Deep FedRec achieved an average accuracy of 81.3% on the MovieLens ML100K dataset, higher than that achieved by simple

FedRec. This clearly indicates that introducing non-linearity in the model proved beneficial.

5.2 Prior Work: Deep Autoencoder based Collaborative Filtering

Introduced by Kuchaiev and Ginsburg [2017], deep autoencoder architecture for recommender system aims at generating missing entries in a rating vector through an autoencoder architecture.

5.2.1 Model Architecture

Fig 5.4 shows the model architecture of the deep autoencoder. The figure is taken from the paper Kuchaiev and Ginsburg [2017]. Architecture is described below-

Encoder It consists of 3 layers of size (512, 512, 1024). The dimension of z is 1024 units. The input layer is of size m, which is equal to number of unique items in dataset.

Decoder It consists of 2 layers of size (512, 512). The output layer is of size m.

Activations The authors use SELU activation function at the output of each layer.

DropOut Layer The authors include a dropout layer at the output of the encoder for regularization. They set a high dropout value of 0.8 on their benchmark model.

5.2.2 Loss Function

The authors use the Masked MSE loss function for training. The loss function does not incorporate loss from those items in the output which were unrated in the input(i.e, their rating value was 0).

$$MMSE = \frac{m_j * (r_j^p - r_j)^2}{\sum_{j=0}^{j=m} m_j} \tag{5.1}$$

where m_j is the boolean whether item j is rated or not, r_j^p and r_j are the predicted and true rating for item j respectively.

5.2.3 Dense Refeeding

The input vector of the network is usually sparse as in our previous discussions, since the user can practically only rate a small fraction of movies of all. The authors thus use dense refeeding, which is the process of performing training steps with the prediction vector of the network as a sample in the training set. Dense refeeding steps-

1. Given input x, compute f(x) in forward pass.
2. Compute loss and perform backpropagation
3. Use f(x) as a training sample to compute f(f(x)).
4. Compute loss over f(f(x)) in which loss is propagated from all items and perform backpropagation
5. Perform steps (3) and (4) again if necessary.

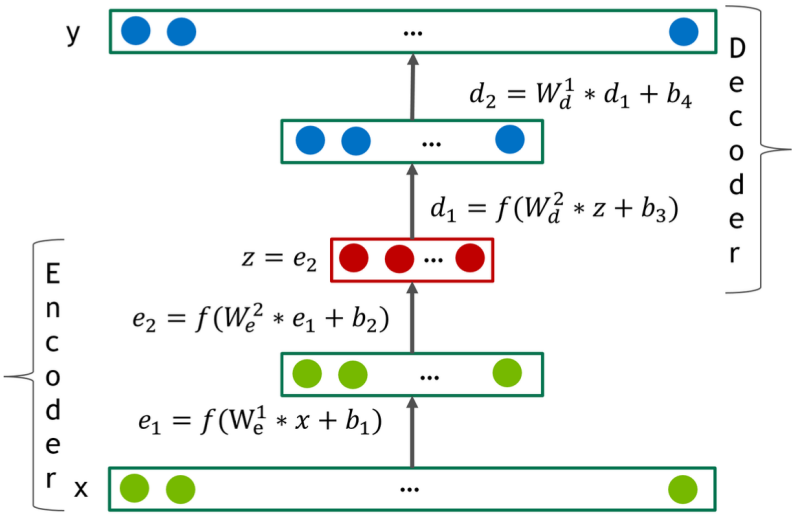

Figure 5.4: Deep Autoencoder Architecture. Ref- Kuchaiev and Ginsburg [2017]

5.3 Introducing Deep Federated AutoEncoder Recommender System

We took inspiration from the autoencode based architecture in Fig. 5.4 to employ it in a federated setting.

5.3.1 User Model

Each user will now have a local autoencoder which will be used to predict ratings. The local user model will only take one rating vector of dimension $1 \times m$, which is stored locally.

The model of our network, described below, is largely inspired from Kuchaiev and Ginsburg [2017]

Encoder It consists of 3 layers of size (32, 64, 64). The dimension of z is 64 units. The input layer is of size m, which is equal to number of unique items in dataset.

Decoder It consists of 2 layers of size (32, 32). The output layer is of size m.

Activations We use SELU activation function at the output of each layer here.

DropOut Layer We use a dropout rate of 0.5 for training.

5.3.2 Gradient flow architecture

The following describes the flow of the gradient through the client-server network. See figure 5.5.

1. For each client i, given sparse local rating vector R_i, compute f(R_i) in forward pass through the autoencoder. Apply dense re-feeding for 2 steps.
2. Loss is computed and gradients are generated.
3. For the decoder, ∇W_d are applied to the decoder locally.
4. The encoder gradients ∇W_e are sent to server for aggregation.

5. Encoder gradients are aggregated and ∇W_e^{global} is sent back to the client for updating the encoder..

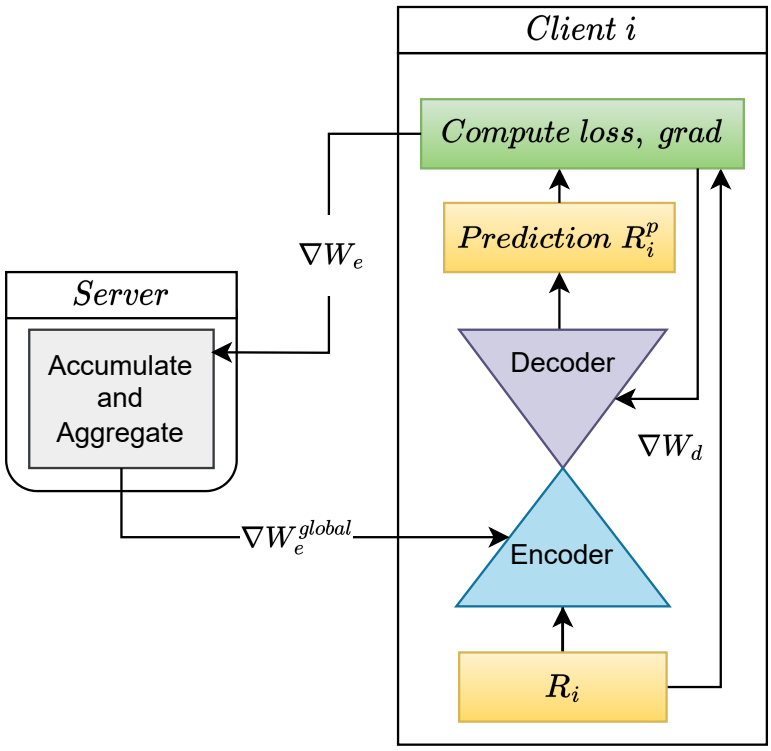

Figure 5.5: Federated Deep Autoencoder Architecture

5.3.3 Loss Function

We use Masked MSE loss function from Eq. 5.1 here as before.

5.3.4 Experiments

The model was implemented using PyTorch and Python.

Dataset The model was trained on the ML100K dataset from MovieLens. The dataset contains ratings from 1000 users on 1700 movies.

Test-Train Split 30% of the rated items were masked from each user's rating vector to create the training vector. The original vector was used to calculate the test accuracy at the end of each epoch.

Table 5.2 summarizes the training hyper-parameters.

Encoder dimensions	$32 \times 64 \times 64$
Decoder dimensions	32×32
Dropout rate	0.5
Encoder learning rate	0.001
Decoder learning rate	0.001
Dense re-feeding	2
Epochs	45
Optimizer	Adam

Table 5.2: Training parameters for Federated Deep Autoencoder

5.3.5 Results

Deep Federated Autoencoder achieved 93.8% accuracy on ML100k dataset, higher than Deep FedRec and simple FedRec. Further, it converged much faster(in 45 epochs) as compared to over 300 epochs for previous approaches.

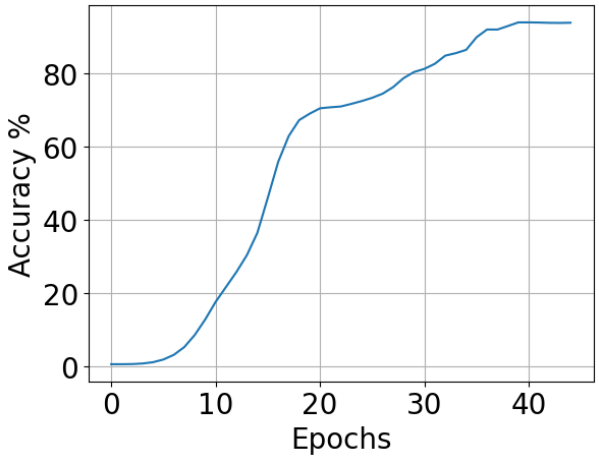

Figure 5.6: Federated Deep Autoencoder Architecture Accuracy vs Epochs

Chapter 6

Conclusions

We thus show 4 progressively advanced techniques to train a federated recommender system. We were able to achieve near state-of-art results while using a federated approach.

By fixing the dataset at ML100k, we were able to fairly compare the performance and efficiency of these algorithms. Federated AutoEncoder gave the highest performance and efficiency of all. Deep FedRec was a close second.

The proposed approaches can be further advanced, such as using Transformers architecture as the user model to train on much larger datasets.

References

Ravikumar Balakrishnan, Tian Li, Tianyi Zhou, Nageen Himayat, Virginia Smith, and Jeff Bilmes. Diverse client selection for federated learning via submodular maximization. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=nwKXyFvaUm`. 10, 11

Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Towards understanding biased client selection in federated learning. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 10351–10375. PMLR, 28–30 Mar 2022. URL `https://proceedings.mlr.press/v151/jee-cho22a.html`. 11

Oleksii Kuchaiev and Boris Ginsburg. Training deep autoencoders for collaborative filtering, 2017. vii, 16, 17, 18

Guanyu Lin, Feng Liang, Weike Pan, and Zhong Ming. Fedrec: Federated recommendation with explicit feedback. *IEEE Intelligent Systems*, 36(5):21–30, 2021. doi: 10.1109/MIS.2020.3017205. 6, 8

Ruisheng Zhang, Qi-dong Liu, Chun-Gui, Jia-Xuan Wei, and Huiyi-Ma. Collaborative filtering for recommender systems. In *2014 Second International Conference on Advanced Cloud and Big Data*, pages 301–308, 2014. doi: 10.1109/CBD.2014.47. 4