Federated Inductive Recommender System

Bachelors of Computer Science

Abdul Rahman
M. Abdur Rahman
Arsalan Tahir

BSCS18022
BSCS18034
BSCS18051

Session: 2018 – 2022

DEPARTMENT OF COMPUTER SCIENCE

INFORMATION TECHNOLOGY UNIVERSITY

LAHORE, PAKISTAN

Federated Inductive Recommender System

A report submitted in partial fulfillment of the requirements for the Degree of Computer Science

Abdul Rahman
M. Abdur Rahman
Arsalan Tahir

Advisor: Dr. Ali Ahmed

Co-Advisor: Usman Anwar

Declaration

This report is a presentation of our original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgment of collaborative research and discussions. I also declare that this work is the result of our own investigations, except where identified by references and free from plagiarism of the work of others.

Signature: ………….…….
Abdul Rahman
Date: ………………..….…

Signature: ………….…….
M. Abdur Rahman
Date: ………………..….…

Signature: ………….…….
Arsalan Tahir
Date: ………………..…....

The undersigned hereby certify that they have read and recommend the thesis entitled “Federated & Inductive Recsys” by Abdul Rahman, M. Abdur Rahman, Arsalan Tahir for the degree of Bachelors of Computer Science.

Dr. Ali Ahmad (ITU), Thesis Supervisor

Usman Anwar, Thesis co-Superviser

Dr. Mohsen Ali (ITU), Committee Member

Dr. Mahboob ur Rahman (ITU), Committee Member

Dr. Arif Mehmood, Chairperson Department of CS

Table of Contents

List of Figures

Abstract

Recent years have seen rapid adaptation of Machine Learning (ML) due to its wide range of applications. The privacy awareness initiatives such as EU General Data Protection Regulation (GDPR), subdued ML to privacy and security assessments. To get a private and secure system, Federated Machine Learning (FL) technique can be used. Federated Learning (FL) is a type of Machine Learning that grants a privacy-driven and decentralized training scheme. It inherently provides a certain degree of privacy by not communicating data directly.
One of the significant aspect of modern web is information filtering. Recommender Systems are used for personalized information filtering and Machine Learning is heavily used in it. To respect users' privacy and create a Recommender System without centralized data, Federated Learning can be used. State-of-the-art works like FedMF and FedGNN presented practical Recommender Systems in federated setting. However, the recommendations learned in these algorithms are transductive i.e., they do not generalize to unseen data. Transductive models require retraining in case new data is encountered. This creates a problem in decentralized training as all data may not be not readily available for retraining. Furthermore, as there are only handful of clients available at a given time period, retraining over a sufficient number of clients can take a long time. To solve this problem, inductive Recommendation Systems can be used. We propose a GNN based inductive Recommender System in federated setting, **FedIGMC**, To the best of our knowledge, this is **first work** that proposes **Inductive Recommender System in Federated Learning** setting.

Chapter 1
Introduction

Machine Learning is applied to broad number of tasks like autonomous vehicles, medical procedures and diagnostics. Machine Learning is also used in some critical scenarios, e.g., autonomous driving, which requires safety measures for preventing accidents. Recent year have seen rising concerns about privacy and user rights. Legal regulations like EU General Data Protection Regulation (GDPR) – European Parliament and Council Regulation No 2016/679 also limited the amount of data tech companies can collect. Problem is that Machine Learning is made possible by the data collected from users. In 2016, Google proposed a way to perform Machine Learning without acquiring data from users called Federated Learning (FL). Federated Learning algorithms train Machine Learning models in decentralized fashion without bringing users' data to their servers. FL growed in popularity due to its broad applicability in creating private and secure systems. Preserving privacy is now the primary concern of FL's development.

The emergence of Youtube, Amazon, Netflix and other e-commerce, online advertisements and entertainment web services made recommender systems (RecSys) a critical part of users' interaction with the internet. Recommender systems rely on Machine Learning and users' data. Concerns regarding privacy created a need to develope a private Recommender System that ensures user's data privacy and security. That system should also be comparable to state-of-the-art centralized Recommender Systems in quality and performance. For this, Federated Learning can be used. In this work, we propose a Federated Learning algorithm for training a Recommender System without collecting user's data. Furthermore, we also try to illustrate why current federated recommendation algorithms are not properly equipped for this task.

1.1 Overview

This work gives a tutorial of Federated Learning (FL), its capabilities and vulnerabilities. Introduction to Recommender Systems and its important techniques. Analysis of current Federated Recommender Systems and their inadequacies. We perform empirical analysis of:

- Federated Learning Inference Attacks
- Federated Recommender system using Matrix Factorization

Lastly, we propose a Federated Recommender System, *Fed-IGMC*, that solves the problems identified in existing Federated Recommender Systems.

1.2 Objectives

The objectives of this work is to give an introduction to Federated Learning and perform brief analysis of its Vulnerabilities and Challenges. Analys the effectiveness of common Federated Learning Defenses. Give an overview of Recommender Systems, its major federated and non-federated techniques. Explain the utility of Federated Learning in privacy concerned society and propose a privacy preserving Federated Recommender System algorithm.

1.3 Limitations and Scope

This work main objective is to study current state of the art Federated Learning specifically in the domain of Recommender systems. This work is just a brief introduction to Federated Learning and Federated Recommender Systems and only discusses the major elements and algorithms of both, the better technique for implementing Recommender Systems in Federated Learning and a proposed solution based on those methods. Other elements of Federated Learning such as its defenses and challenges are discussed but defining the best approach in those tasks is beyond the scope of this work.

1.4 Thesis Outline

We organize the rest of this dissertation as follows. Chapter 2: An introduction and tutorial of Federated Learning. Chapter 3: Federated Learning Vulnerabilities. Chapter 4: Overview of Federated Learning Defenses. Chapter 5: A brief introduction to Recommender Systems . Chapter 6: Federated Recommender Systems. Chapter **??**: Proposed Solution: Fed-IGMC. Chapter 8: Conclusion and Discussion.

Chapter 2
Federated Learning

In 2016, Google presented a novel decentralized machine learning scheme, Federated Learning (FL) [1], that enabled training a single global model on many edge devices without exposing the data. Recent privacy concerns also boosted the adaptation of Federated Learning and privacy preservance became the main objective for developments in Federated Learning techniques. Federated Learning trains model collaboratively so it is also known as Collaborative Learning. One of the goals of Federated Learning is to share minimum amount of information possible, so participating devices only share gradients found in training for progression of global model.

2.1 Machine Learning (ML)

Federated Learning (FL) is decentralized Machine Learning (ML). To understand FL better, following is a short review of ML. ML is an automated procedure for learning mappings of inputs to outputs based on the data present. These mappings are usually called models and their task is to predict output for unseen inputs. Models update their structure by iteratively training on data to minimize loss. ML can be supervised, unsupervised, semi-supervised, and reinforcement based on the type of data that is available [2]. In supervised learning, the goal is to perform classification (map an input to a discrete output) or regression (map an input to a continuous output). The model uses a dataset, a collection of input and output pairs $(xi, yi)_{i=1}^{N}$, where N is the size of the dataset for Learning. In classification problem, each sample x_i, called a feature vector, has a corresponding label output y_i belonging to class $c_k : C = \{c1, c_2, c_3, ..., c_n\}$. Each element or feature of the feature vector $x_i^{(j)}$ $(j = 1, 2, 3, ...)$ describes an attribute of the sample. The goal of supervised learning is to create a mapping or model that can infer a label y for unseen feature vector x as input. A model f takes a feature vector x as input for predicting its label y such that $y \leftarrow f(x)$. The model f consists of weight vectors $\theta_{i=1}^{M} = \Theta$ which are optimized in training. Θ is optimized by minimizing a loss function $L(f(x))$ iteratively $argmin_{\Theta} L(f(x))$

2.1.1 Federated Learning

Federated Learning (FL) or **Collaborative Learning** or **FedML** (Federated Machine Learning) is performed on distributed clients hosting disjoint datasets. To begin a ML model is selected which is to be trained, by each client independently on their local dataset. A copy of global model W_t^{global}, at round t, is sent to each client by an aggregator. The training hyper-parameters i.e., batch size, learning rate, number of epochs are all agreed upon. A client i participating in training minimizes its

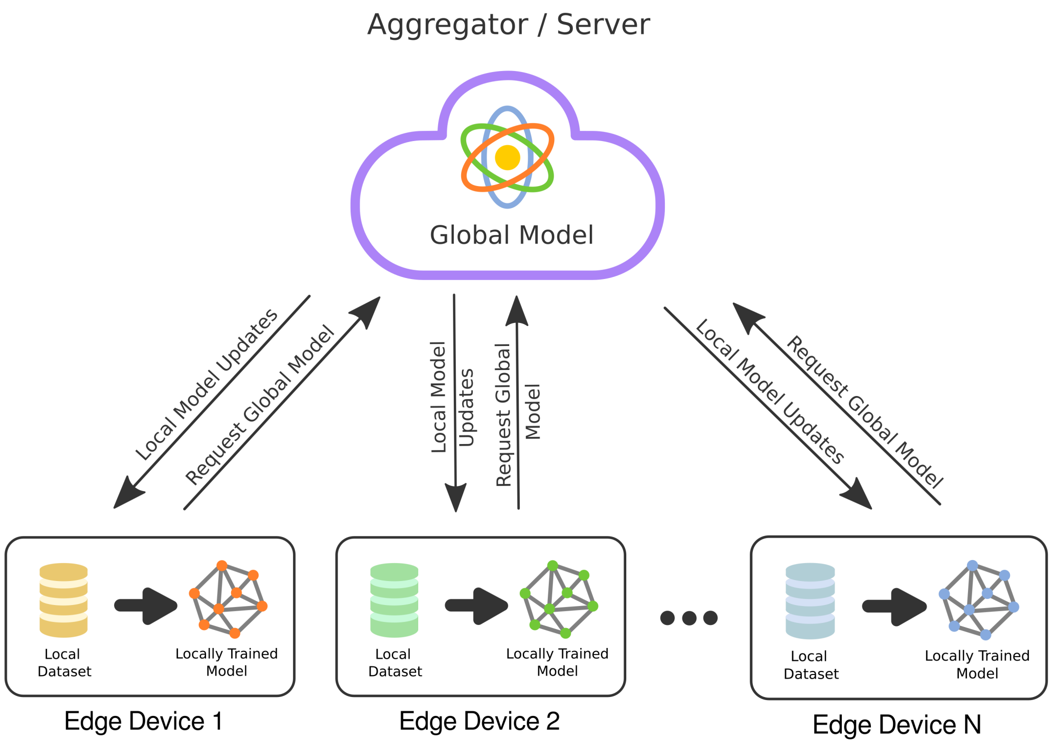


Figure 2.1: Federated Machine Learning General Setup.

local loss function $argmin_{\Theta} L(f_i(x))$ on its recieved copy of the model using its local dataset. After training is complete, each participating client i finds a new local model W^i_{t+1}. These local models will most likely be different from other participants as the datasets of participants were disjoint. The gradients i.e. the difference $W^{global}_t - W^i_{t+1}$, are sent by each participant to the aggregator. The aggregator combines all the participants' gradients to produce a new global model W^{global}_{t+1} which is communicated back to all clients. This process is performed multiple times until certain termination criteria is met. Each iteration of this process is called a training round. A single training round can have multiple local epochs [3].

Aggregator can combine gradients in various ways, one of them is taking weighted average based on the amount of data each participating client was holding, this averaging technique is *FedAvg (Federated Averaging)*.
The aggregator can be a centralized server connected to all the clients. Aggregator can also be removed and aggregation process can be performed among the clients to achieve a fully decentralized or peer-to-peer (P2P) architecture [4].

2.2 Federated Learning vs Distributed Learning

Idea of training a single model on multiple devices is not new, prior to FL, **Distributed Learning** has been used for this. The key difference between FL and Distributed Learning lies in the assumptions on the local datasets. Distributed Learning aims to increase computing power by parallelization of computing power of multiple machines where as FL aims to train on heterogeneous datasets while preserving privacy. In Distributed Learning, the datasets are i.i.d. (independent and identically distributed) i.e., each data-point is equally likely to be sampled hence participants have same data distribution. *Data-point* is a unit of information e.g. a row of a table. Moreover, datasets roughly have the same size. Both of these assumptions don't hold in FL. Moreover, clients involved in FL are usually edge devices (smartphones and IoT devices) with little data, constrained bandwidth and limited availability whereas Distributed Learning usually involves data centers with continuous availability, higher communication bandwidth and huge data. In distributed learning, data is mostly centrally available and is divided on purpose, so different pre-processing and shuffling methods can be applied to make data i.i.d [5].

2.3 Federated Learning Algorithm

Any Federated Learning algorithm can be divided into three parts: Model architecture: The model used in training, optimization methods and hyper-parameters, Aggregation algorithm: The technique which is used by the aggregator/server to combine results from training participants to create new global model. Compression and Obfuscation: This is any add-on to the algorithm to make it more secure or communication efficient and more. All of these parts are tightly coupled.

2.3.1 Model Architecture

As for the model architecture, any Machine Learning model can be used but it should be kept in mind that data is sparse, computational power and communication bandwidth of edge devices is small. In usual setting, the hyper-parameters of model like batch-size, number of epochs, learning rate, optimization technique are decided globally by aggregator. Though in some algorithms, these hyper-parameters might be individual and decided by each client locally. Moreover, clients may also freeze few layers of their model in global training and train them locally to get personalized results. Finally, based on the problem being tackled, all of these functionalities can be fine-tuned to produce best results.

2.3.2 Aggregation Algorithm

Aggregation algorithm is used to combine results from all participants. There are few things to consider when training in Federated Learning, how many clients to involve in each training round, should all clients be selected for training? Should hyper-parameters be global? How to combine participants results into single consolidated model? All of these problems are tackled by the aggregation algorithm. The most basic aggregation algorithm is based on Gradient Descent. In the beginning of each round, aggregator selects a single client at random from the pool of available clients and sends it the global model. The participating client retrieves the model, performs gradient descent on its complete local data, minimizes loss and sends gradients back to the aggregator. The aggregator sends this global model to every client. The participating client only performs single epoch of training with a single batch encompassing all of its data. This technique can be improved in the following way.

Instead of taking a single client, a fraction of clients is selected for training and then their results are combined i.e. averaged to produce a new global model. This is **FedSGD**, the baseline Federated Learning algorithm. The approach of selecting a fraction k of all clients outperforms the state-of-the-art distributed learning asynchronous techniques [6].

FedAvg $(FederatedAveraging)$, introduces more than one client in each round, performs multiple epochs each round on every participant, and local data is divided into mini-batches. At the end of each round, all the participating clients' results are combined in a single result using weighted average based on the amount of data is present on each participant. The batch size, learning rate and number of epochs are communicated by the aggregator to all participants. Usually 0.1 fraction of total clients are included in a round, they are selected randomly. Note not all clients will be available for training at a given time anyway and using a factor gives better results than training on all clients. All participants are to perform the training and communicate their results back to the aggregator/server, if a participant fails to complete training due to any reason in the allocated time then it gets dropped i.e., its contributions are not be considered by the aggregator. The dataset size of each participant is also communicated to the server for weighted averaging. FedAvg was introduced by Google and it is the de-facto standard of Federated Learning aggregation [5]. Basic $FedAvg$ algorithm is given below:

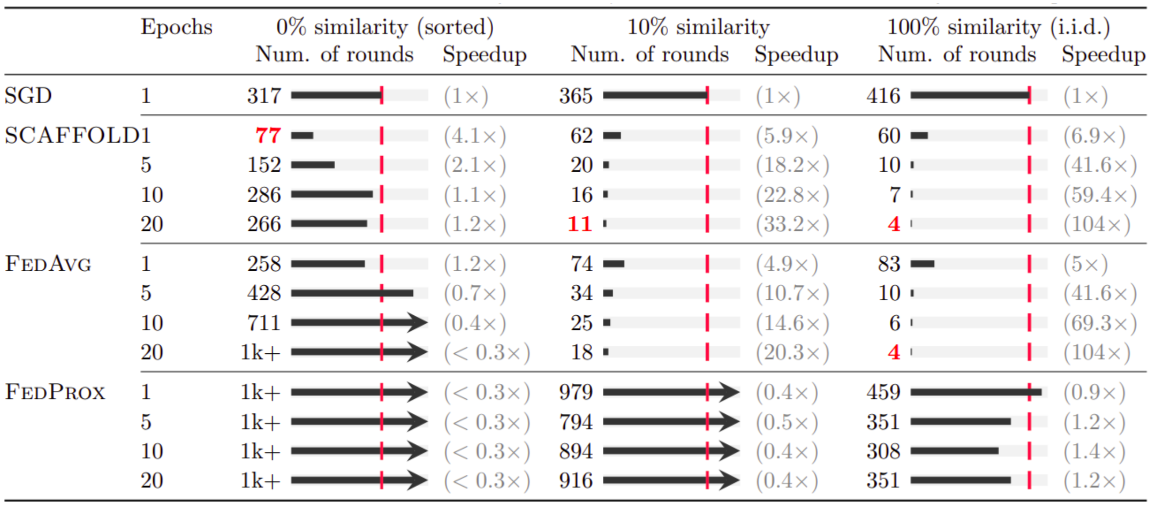
Algorithm 1 *Federated Averaging (FedAvg) Algorithm.* K clients are indexed by k; B is minibatch size, E is number of epochs, η is learning rate

```
Server executes:
initialize $w_0$                                        ▷ $w_0$ are initial weights
for each round $t = 1, 2, ...$ do
    $m \leftarrow max(C.K, 1)$                           ▷ $C$ fraction of clients
                                                        ▷ $K$ number of clients
    $S_t \leftarrow$ (random set of $m$ clients)
    for each client $k \in S_t$ in parallel do
        $w_{t+1}^k \leftarrow ClientUpdate(k, w)_t$
    end for
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$                  ▷ $n_k$ client $k$ dataset size
                            ▷ $n$ sum of all participating clients dataset size
end for

function ClientUpdate($k, w$)                           ▷ Run on clients in parallel
    $\beta \leftarrow split\ \ Data_k$ into batches of size B
    for each local epoch $i$ in $1$ to $E$ do
        for batch $b \in \beta$ do
            $w \leftarrow w - \eta \nabla l(w; b)$
        end for
    end for
    return $w$ to server
end function
```

Federated Averaging technique also works in case of non-i.i.d. datasets i.e. datasets with different distributions, though the results may not be as good as they are for i.i.d. datasets. FedAvg has some restrictions e.g. all participants must use same batch size, same optimization algorithm, same learning rate and perform predefined number of epochs. If a participant is not able to compute the complete output, that participant is dropped i.e. its incomplete results are not used in aggregation. Moreover, although FL trains well on non-i.i.d. data but it can break in case of extreme variance in data distribution or noise. Since model is trained by clients, so if a certain client is adding noisy data there is no way for FedAvg to identify or avoid it (in its out-of-the-box form). Moreover, some scenarios require more sophisticated algorithms that can work with encryption and/or compression. All of these functionalities can be added into FedAvg by making slight changes.

There are many aggregation algorithms based off FedAvg e.g. FedProx, SCAFFOLD, FedOpt. **FedProx** introduces technique to incorporate less computationally efficient participants which were being dropped in FedAvg. The authors of FedProx claim that it tackles the problem of statistical and system heterogeneity better compared to FedAvg. In FedProx, participants can have locally decided optimization method, though learning rate is same for all. Participants which are not able to complete all epochs, get their incomplete results incorporated in the final output after normalization of those results by some factor [7]. This factor can be decided based on how much progress was made. This allows for better inclusion of clients which means model gets trained on more diverse data hence better accuracy. FedProx, though, it might provide good results in case of non i.i.d. data, but in case of i.i.d. data, it produces far worse results than FedAvg or FedSGD even. **SCAFFOLD** (Stochastic Controlled Averaging Algorithm) provides a more practical algorithm to address statistical heterogeneity. In SCAFFOLD, each client maintains a local control variate and there is a universal global variate as well, these are used to nudge the local updates in the direction of the global model hence large divergences are avoided, this in turn helps in early convergence of model on i.i.d. data as well as non-i.i.d. data [8].



	Epochs	0% similarity (sorted)		10% similarity		100% similarity (i.i.d.)	
		Num. of rounds	Speedup	Num. of rounds	Speedup	Num. of rounds	Speedup
SGD	1	317	(1×)	365	(1×)	416	(1×)
SCAFFOLD	1	**77**	(4.1×)	62	(5.9×)	60	(6.9×)
	5	152	(2.1×)	20	(18.2×)	10	(41.6×)
	10	286	(1.1×)	16	(22.8×)	7	(59.4×)
	20	266	(1.2×)	**11**	(33.2×)	**4**	(104×)
FedAvg	1	258	(1.2×)	74	(4.9×)	83	(5×)
	5	428	(0.7×)	34	(10.7×)	10	(41.6×)
	10	711	(0.4×)	25	(14.6×)	6	(69.3×)
	20	1k+	(< 0.3×)	18	(20.3×)	**4**	(104×)
FedProx	1	1k+	(< 0.3×)	979	(0.4×)	459	(0.9×)
	5	1k+	(< 0.3×)	794	(0.5×)	351	(1.2×)
	10	1k+	(< 0.3×)	894	(0.4×)	308	(1.4×)
	20	1k+	(< 0.3×)	916	(0.4×)	351	(1.2×)

Figure 2.2: FL Aggregation Algorithms Comparison. Reprinted from [8]

FedMA proposes an algorithm to generate a normalized form of locally trained neural networks by making them permutation invariant [9]. Since there could be many combination of weights that can map X to y, locally trained models may have different forms. If weights of all participants are averaged and they had different permutations of the same model then the resultant model will not produce good results. FedMA uses numerical approximation methods like Newton's Method to find an optimal weights representation. This approach produces better results, however it is slow and complex.

2.3.3 Compression and Obfuscation

As we will see in Chapter 3, Federated Learning in its out-of-the-box form is not sufficient. Techniques like compression, dimensionality reduction may also be needed to increase communication efficiency. For obfuscation, techniques like Differential Privacy (DP), Cryptography may be used [Refer Chapter 4 to learn more]. Usually obfuscation comes at the cost of accuracy or communication overhead, so a balanced approach is necessary to develop an efficient system. For example, in [10], communication is made efficient by only communicating the gradients which are more than a certain threshold, this technique helps in decreasing communication times. Which means more rounds can be performed with less number of epochs that can help in training the model better. This approach can also help against inference time attacks [Refer to Chapter 3 to learn more].

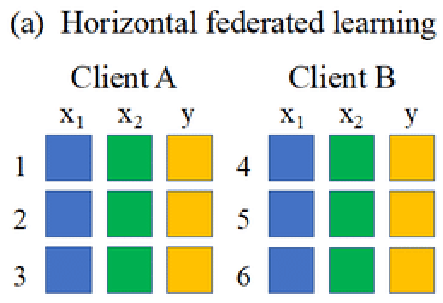


Figure 2.3: Horizontal FL, Reprinted from [12].

2.4 Federated Learning Types w.r.t. Data-points Distribution

Mostly in FL, data present on clients will be sparse, unbalanced and of different sizes relative to eachother. Data-points can overlap and they may also differ in labels from client to client as well. These are some common challenges related to statistics of data in FL and are usually termed as statistical heterogeneity of data. Section 2.5 discusses them in detail.
Apart from this, based on how data-points are present on clients, FL can be classified into *Horizontal FL* (HFL), *Vertical FL* (VFL), and *Federated Transfer Learning* (FTL).

2.4.1 HFL: Horizontal Federated Learning

Each client in **HFL** has same set of features while the data-points are unique. Consider two clients i, j hosting data privately. The datasets of i and j are horizontally partitioned when they share same feature space $X_F^i = X_F^j$ whereas the data-points client i and j host are disjoint $X_r^i \neq X_r^j$, here a single data-point is denoted by r (record). Without the loss of generality, data-points can be represented as rows of a table and its fields as features as shown in Figure 2.3. If each record (data-point) is unique and covers all the features $(x1, x2)$ then HFL will be used for training on such data. Data is distributed among many clients but that distribution is in terms of records, note the ID of each record is unique irrespective of client in the Figure 2.3 [11]. Most common distribution of data is horizontal distribution.

2.4.2 VFL: Vertical Federated Learning

VFL is used when the data-points are partitioned such that clients can have same data-points but distinct features. Consider two clients i, j hosting data privately. The datasets of i and j are vertically distributed when they share distinct feature space $X_{F\,F}^{i\,j}$ but same datapoints/records $X_r^i = X_r^j$. This

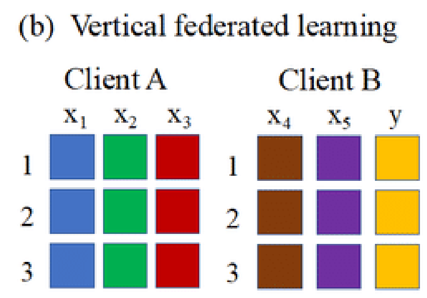


Figure 2.4: Vertical FL, Reprinted from [12].

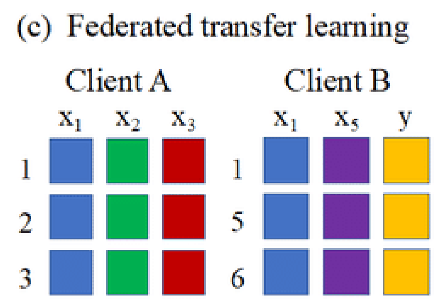


Figure 2.5: Federated Transfer Learning, Reprinted from [12].

represents the situation where same records (datapoints) are present among various clients but those clients host distinct features of those records [13] as shown in Figure 2.4. Note that the ID of records are same but features of those records are different for both of the clients i.e., x_1, x_2, x_3 for *Client A*, x_4, x_5, y for *Client B*.

2.4.3 FTL: Federated Transfer Learning

FTL is used to perform Transfer Learning in FL, a technique that maps a model trained on a certain domain and uses it to perform predictions on a target domain. Feature space and dataset of both source and target can be different with a little overlap between them (as shown in Figure 2.5). This allows using a model which is trained to solve a certain problem and use it to solve a different problem. The overall model quality depends on the how well both problems co-relate. Pre-trained models can enhance the overall quality of the training [14].

2.5 Federated Learning Types w.r.t. Client Capabilities

FL's target is to perform training on decentralized data privately. Till now, we have talked about the cases where clients are edge devices like smartphones or IoT devices with limited computational and communication power, but FL can be used with high power machines like data centers machines. Data centers might want to train a global model but without sharing any data with each-other e.g. multiple hospitals might want to train a global model but they cannot share patients' data with each-other. Based on the client's capabilities, FL can be divided into two types: Cross-silo and Cross-device [15].

2.5.1 Cross-silo Federated Learning

In Cross-silo setting, the network contains a few devices with high computation power e.g., data centers. Cross-silo architectures can support more communication bandwidth, with better availability and more data as compared to cross-device settings.

2.5.2 Cross-device Federated Transfer Learning

The network contains many devices having low computational power in Cross-device setting, clients are usually IoT devices or smartphones.
Note: Federated Recommender systems have horizontal data distribution and mostly used in edge devices hence data is collected and present on low power devices.

2.6 Federated Learning Challenges

There are four core challenges in FL; communication efficiency, system heterogeneity, statistical heterogeneity and, privacy preservance. Following is a brief description of each:

2.6.1 Communication Efficiency

Communication efficiency is important in FL as edge devices have low network bandwidth, which is low for download and even lower for upload. So, training and communication are to be carried out at a certain pace or some slow participants will be left out most of the time. To train model better, maximum inclusion is necessary, so it is highly undesirable that a certain group of clients are always left out due to limited bandwidth. Communication efficiency can be increased by using smaller models, lesser rounds, more epochs, compression but all of these come at some cost like model quality and/or computational load. Moreover, its likely that devices with limited communication bandwidth will have limited computational power as well. Furthermore, model training is done when edge-devices are in idle state, charging and connected to WiFi. This largely limits the availability of devices at a single point in time. Factors like location can have an impact on the availability of devices i.e., phones from specific locations may become available at same time as they may get in idle, charging state and connected to WiFi in night. It is quite possible that the same devices become available for training every time and a large chunk never becomes. These factors can have adverse affect on model quality.

2.6.2 System heterogeneity

System heterogeneity refers to the difference between the devices that are being used in the training, such as devices' processing power. If the model is complex and large number of training epochs are being performed, then some devices with lower processing power may never be able to complete the training in allocated time by the aggregator. So, few people with higher computing power will dominate and set the direction of model. Also, output produced by a certain device A may vary from the output produced by another device B with the same model and same data due to underlying architectures of devices and computation precisions [16].

2.6.3 Statistical Heterogeneity

General setting of Cross-device FL has large number of clients with data largely spread out in uneven amounts and different data distributions. It is likely that clients available at a particular time might have similar data characteristics due to same geo-region as location is somewhat correlated to data distribution. It is also possible that devices having certain kind of data with distribution A are available at a certain time and devices with different type of data distribution B are available at another time with no overlap in between. In such case the model will be going back and forth between two targets and will train poorly. This and other similar factors may cause model to diverge instead of converging. The statistical heterogeneity of the data is a big challenge in finding a good global model. Data heterogeneity can occur in many forms e.g. distributions may be different, the local distributions may change overtime, the data itself may be disjoint, the data might be same but have reverse labels [17].

A workaround of this problem is to make the models personalized, i.e. a part of the model is made global and some part is left local. Each participant only communicates the gradients from its global part and not from its local part. Moreover, basic aggregation algorithms like $FedAvg$ work well against non-i.i.d. data in practice.

2.6.4 Privacy

The basic idea behind FL is minimal information should be communicated to aggregator hence only gradients are sent. This seemingly makes the system private and secure. Due to the rising concerns about privacy, this promising private learning technique has been put to thorough assessments. These evaluations have found that privacy leaks can occur in different phases of FL. Depending upon the amount of information that is available to the adversary and adversary's role in the system, different attacks can be performed on the system. Chapter 3 discusses the privacy Vulnerabilities in FL and Chapter 4 discusses the possible Defenses.

Chapter 3
Federated Learning Vulnerability Analysis

Though data never leaves the device, still FL can be vulnerable to attacks. Attacks can be performed by clients as well as the aggregator/server. The goal of privacy preserving training is to keep a client's private data secure from other clients and server while also ensuring the quality of model being produced.

3.1 Vulnerabilities inherited from Machine Learning

Recent works have shown many FL vulnerabilities [18]. Attacks in training and inference phases can lead to information leakage and/or system failure. FL inherits vulnerabilities from Deep Learning which is a branch of Machine Learning that is mostly used these days. But Deep Learning's main goal is not privacy preservance. Meanwhile newer vulnerabilities are introduced as well due to the FL's architecture [18]. To begin, we divide vulnerabilities in two **threat models**, white box and black box.
In **black box** model, an adversary can generate output $f(x)$ for particular input x without seeing any internal computations.
In **white box** model, the adversary has internal information about Machine Learning model like model architecture, communication hyper-parameters and may have access to gradients communicated by the clients to aggregator.
The goal of the attack could be to degrade the quality of the model i.e., **untargeted attack** or to generate some specific wrong output on specific input i.e. **targeted attack**.

There are two phases of model development: *training* i.e., optimizing weights so they map inputs to outputs, and *inference* i.e., when trained model is used to predict. Attacks are possible in both of these phases.

3.2 Training Time Attacks

Training attacks can be classified into poisoning attacks and free-rider attacks. In **Poisoning attacks**, a client performs a modification to the dataset or weights to degrade the model performance.
Data Poisoning attack modifies data to diverge the model. The data may be modified by inverting the output values. Generative Adversarial Networks (GANs) i.e., an unsupervised deep learning technique for generative modelling, can also be used to generate adversarial data. This attack is not so effective on its own as updates from multiple clients are averaged so input of a specific client will get

diminished.
Another type is **model poisoning** in which adversary participating in the training modifies resultant gradients directly and makes attack more efficient by enlarging its contribution i.e., aggregation algorithms use weighted averaging based on amount of data present on a particular client so that can be exploited by the client. This exploitation is called **boosting**. Most effective model poisoning technique is Sybil attack, i.e. adversary makes multiple clients all of which have no data but they send fake updates to aggregator to degrade the model.
Backdoor poisoning is data poisoning on chosen labels (outputs) and training accurately on the rest of the labels. This is effective with boosting. Data can also be modified instead of labels in backdoor poisoning. Backdoor poisoning can be implemented by just modifying a single input element e.g., pixel in case of images. This kind of attack cannot be detected easily.
In **free-rider attacks**, random values are sent as gradient updates along with boosting, this degrades the convergence rate of model. Degradation could be upto 50% even in state-of-the-art models.

There are few *defenses* against these attacks, first is to limit the contribution by any single participant to stop boosting, ignoring any participant that claims to have dataset of size above certain threshold. To prevent poisoning, current model can be compared with the newly received models from participants and if any of the received models has high co-variance with current model then ignore that participants contribution in training.

3.3 Inference Time Attacks

Evasion attack aims to deceive the model. It evades correct prediction or classification at the time of inference. A carefully crafted noise is added to original input x such that $f(x + noise) \neq y$ where as $f(x) = y$. This kind of attacks are useful for evasion like avoiding face detection.
Other inference time attacks may try to extract information by performing inference on the model. Inference attacks can be classified into four types: *membership inference, property inference, model inversion, model extraction attacks.*
Membership inference is used to infer whether the model has been trained on some data X, it can also suggest which client owns the data X.
Property inference can infer if data X on which model is updated had property p. This is done by learning a binary classifier by giving it two snapshots of updates one having property p and one without it. So, when any update is given to that binary classifier, it can tell if the data from which we achieved the update had property p or not.
Model Inversion attacks predict data using the label and/or gradients. Model inversion attacks can be performed by white box as well as black box threat models. Pixel-wise accurate images and token accurate text have also been achieved using this technique. Section 3.4 and 3.5 demonstrate practi-

cally possible model inversion attack and perform empirical analysis of algorithms that can be used for it.

Model Extraction attacks are used in black box model where we infer the model f structure by analyzing its output for specific inputs and trying to mirror it in a new model $f^{'}$ using an analytical approach, so that $f(x) \approx f^{'}(x)$. The models are not architecturally same but they make same predictions.

3.4 A Demonstration of Model Inversion Attack

Gradient leakage or Model Inversion is a general problem of Machine Learning and not exclusive to Federated Learning. Any gradient sharing scheme is prone to *gradient leakage* attacks where gradients can be exploited to approximate inputs. Generally, a client trains local model on its data (an image in this demonstration) and shares it with the aggregator/server (adversary) that is designated to combining the local updates and producing new global model.

Here we provide empirical analysis of *Model Inversion Attack* using the work 'Deep Leakage from Gradients (DLG)' [19] implementation. In DLG, an adversarial server aims to attack a specific user in order to approximate its local data that was used in training.

For any FL scheme having a ML model $F()$ with learnable parameters W and a central server that gets gradient updates ∇W from a client with respect to client's data (a pair of inputs and labels), the server can obtain the training data (inputs and labels) of the client reversely from its gradients. During training, client i has a copy of global model with weights W_t, after training the model on local data, client gets a new set of weights W^i_{t+1} such that $W_t - W^i_{t+1} = \nabla W^i_{t+1}$ where ∇W^i_{t+1} are the gradient updates of weights of round $t+1$. Server chooses a target client. Server starts to receives gradient updates ∇W from the target client. The server does not aggregate the updates from target to the global model. Instead to initiate the attack, it initializes dummy/random input data x' and label y'. It performs inference/prediction on the dummy input x' and apply loss using dummy label y' to achieve dummy gradients ∇W'. Now the server optimizes dummy inputs x' and dummy labels y' to minimize distance between dummy gradients and client's gradients. When dummy gradients and client's gradients match, the optimized input and label represent the actual data of client that was used in training. Following objective function is minimized:

$$\mathbf{x}'^{*}, \mathbf{y}'^{*} = \underset{\mathbf{x}', \mathbf{y}'}{\arg\min} ||\nabla W' - \nabla W||^2 = \underset{\mathbf{x}', \mathbf{y}'}{\arg\min} || \frac{\partial \ell(F(\mathbf{x}', W), \mathbf{y}')}{\partial W} - \nabla W||^2$$

Figure 3.1 shows an illustration of the attack. Normal participant trains model F() on the cat image x with label y and produces ∇W. The malicious attacker which only has access to ∇W of the participant initialize model with dummy input and label and produces $\nabla W'$. For every iteration,

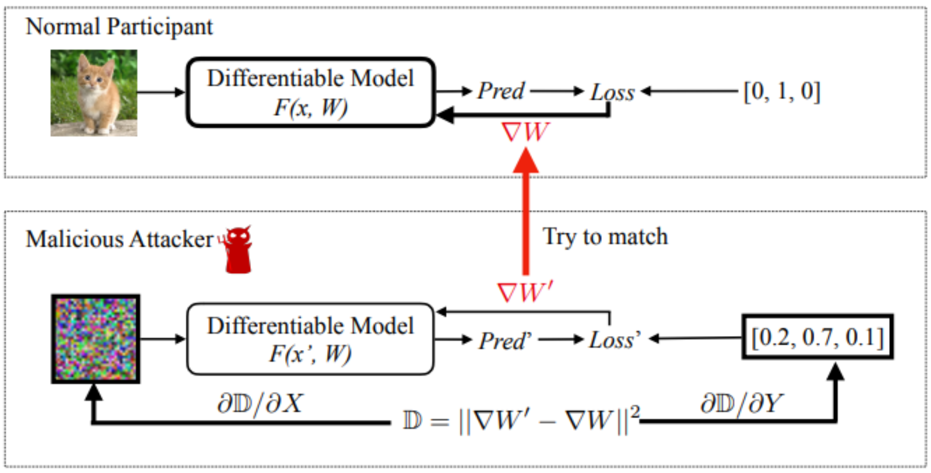


Figure 3.1: Overview of DLG Algorithm. Reprinted from Deep Leakage from Gradients, NeurIps, (2019).

attacker gets ∇W of the user and minimizes objective function D with respect to X, Y, updates X, Y and sends ∇W back to the participant.

Algorithm 2 DLG

Input: $F(x : W)$: Differentiable machine learning model; W: parameter weights; ∇W: gradients calculated by training data

Output: private training data x, y

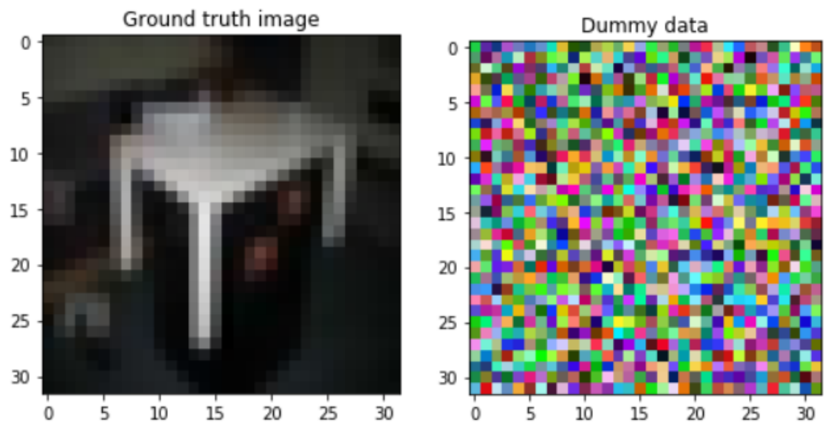
```
1: function PROCEDUREDLG($F, W, \nabla W$)
2:     $x'_1 \leftarrow \mathcal{N}(0,1), y'_1 \leftarrow \mathcal{N}(0,1)$        ▷ Initialize dummy inputs and labels
3:     for $i \leftarrow 1 ton$ do
4:         $\nabla W'_i \leftarrow \partial l(F(x'_i, W_t), y'_i)/\partial W_t$        ▷ compute dummy gradients
           $D_i \leftarrow ||\nabla W'_i - \nabla W||^2$
           $x'_{i+1} \leftarrow x'_i - \eta \nabla'_{xi} D_i, y'_{i+1} \leftarrow y'_i - \eta \nabla'_{yi} D_i$        ▷ update data o match gradient
5:     end for
6: return: $x'_{n+1}, y'_{n+1}$
7: end function
```

Algorithm of adversarial server is shown above. In line 2, dummy input x and dummy label y is initialized. Then for every iteration, ∇W of the client is fetched (line 4), and x', y' are updated/optimized.

3.5 Empirical Analysis: Model Inference Attack

DLG implementation is tested on images from $CIFAR$ dataset. For fast regeneration of inputs, Batch size of 1 is used. A random image is selected as input for this attack's demonstration.

The image selected has white table in it and its label is 84 in $CIFAR$ (shown in Figure 3.2 (a)). Dummy image and dummy label is initialized, image is shown in Figure 3.2 (b). Dummy label is 64.



(a) Dummy Image (b) Training Image

Figure 3.2: DLG Attack Initialization.

Figure 3.3 shows regeneration of training image after every 10 rounds of server aggregation, as number of rounds increase, the result gets better and better.

Code is available at https://github.com/arahman01/FYP.git

Limitations This approach uses a *Batch size* of 1 i.e., the local participant only trains model on a single image. In later demonstration, the batch size is increased to few images but computational time increases exponentially and the algorithm is too slow to converge. A way to overcome this issue is to optimize a single image in the batch at once. This leads to convergence quicker than trying to optimize all images together but this only works till batch size of 8 as noted by the authors. An increase in batch size exponentially increases the number of iterations required for the DLG attack model to converge. Moreover, several communication rounds are needed with same client so a special version of model can be converged on that client's data. Another limitation is amount of the detail input has i.e., resolution of image in this demonstration. DLG currently only works for image resolution upto 64×64. DLG would not work if training scheme uses *Differential Privacy* and/or *Compression*. Using mini-batches would render the attack useless as well.

Recent works in Model Inference Attacks propose better reconstruction techniques which are immune

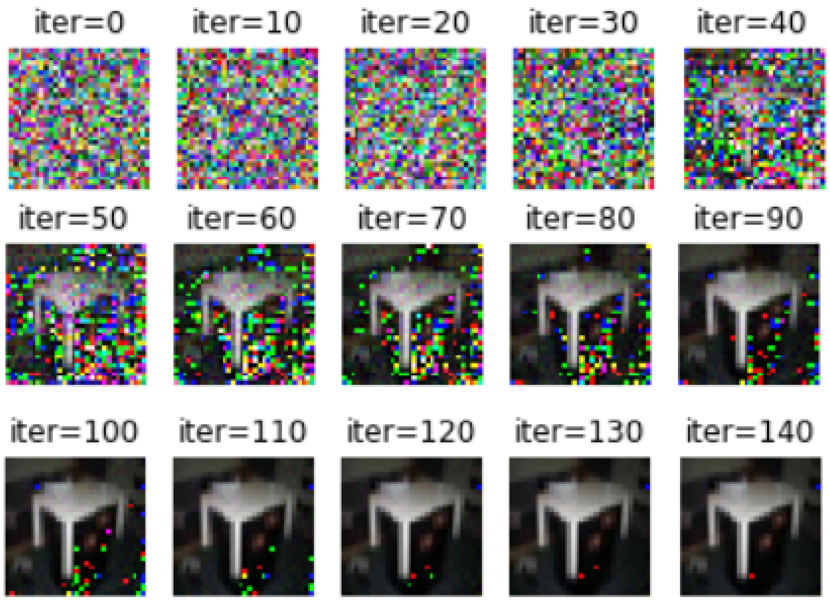


Figure 3.3: DLG: Image Regeneration Iterations

to Differential Privacy and can generate large batches effectively [20] [21].

3.6 Privacy Breach Study: Man in the Middle Attack

Model Inference attack poses a serious privacy challenge if communication's security gets compromised. In such case, an adversary can perform *Man-In-The-Middle (MITM)* attack and acquire gradients of the client. These gradients then can be used to perform model inference attack. *MITM* attack or spoofing general setup consists of a victim (V), server (S) and an adversary (M). In federated setting, server S is aggregator, victim V is an active client and adversary is another client that compromised the communication between server and victim client.

Figure 3.4 explains the procedure of *MITM* attack.

1. The adversary aims to get client V's local data. The adversary joins the network as a client. It intercepts the communication between client V and server S and breaks the communication channel A.
2. Adversary poses as S (server) to the V (victim) and as V (victim) to the S (server) and establishes communication channels B with V and C with server.

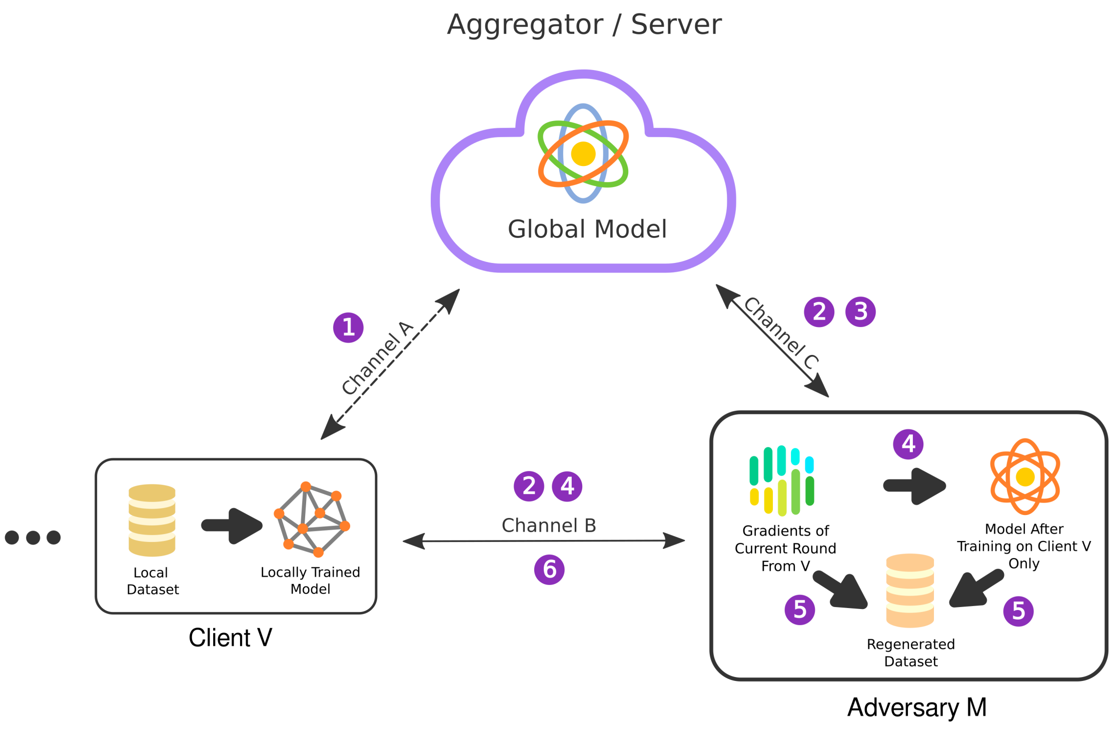


Figure 3.4: Man in the Middle Attack.

3. Adversary gets model from server or creates its own.
4. Adversary M instructs client V to perform training by sending V its model. V thinking M is server, performs training and sends gradient updates back to adversary M. Adversary uses these gradients to update its model.
5. Adversary uses gradients from V and its model to regenerate V's data using dummy inputs and gradients. It optimizes inputs such that dummy gradients obtained from those inputs match V's gradients.
6. Adversary performs multiple rounds to accurately regenerate data.

Adversary may send client V's gradients or gibberish back to the server S if/when server asks for updates from adversary M thinking its client V.

This shows the effectiveness of gradient leaks for data regeneration. And how a security leak can jeopardize privacy as well.

Chapter 4
Privacy Preserving Techniques and Defenses in Federated Learning

From Chapter 3 it is evident that Federated Learning alone in its out-of-the-box form is not sufficient for guaranteeing privacy as sensitive information may get leaked indirectly. For our aid, we can deploy various methods like encryption, noise, filtering to better equip Federated Learning. FL algorithm should be equipped properly to handle adversarial activities from both, client or server/aggregator. Following are some common techniques that are used to prevent adversarial attacks in FL.

4.1 Secure Aggregation using SMC: Secure Multi-Party Computation

Secure Aggregation is a process of only revealing the aggregated model to the server/aggregator and intermediate local models are obfuscated so server cannot extract information from them. This makes the adversary attacks less effective as an adversary cannot directly pin-point exact inputs of any specific participant. Secure Aggregation can be implemented using SMC techniques like Shamir Secret Sharing and Homomorphic Encryption.

0SMC is subfield of cryptography which devise methods for participants to jointly compute a function while keeping their inputs private. In traditional cryptography, the goal is to make communication or storage secure such that adversaries not participating in the system cannot decrypt or understand the messages. Whereas, in Secure Multi-Party Computation (SMC), a function is computed by combining participants data while participant's messages are protected from eachother [6]. Note that these cryptographic techniques require significant communication resources and are computationally expensive.

4.1.1 Secret Sharing

Secret Sharing: Secret sharing techniques like Shamir Secret sharing allows a n number of participants share a secret which requires atleast any t number of participants to reveal. So, any random $t - 1$ or less number of participants cannot break the secret. Secret sharing in combination with digital signatures and third party key generator servers can be used in Federated Learning for Secure Aggregation. Secure Aggregation techniques do exist that can work without third party server aid but most common approaches use third-party server.

4.1.2 HE: Homomorphic Encryption

Homomorphic encryption is a common cryptographic defense used in FL. HE is a special type of encryption in which mathematical operators over encrypted data and non-encrypted data result the same i.e., two encrypted numbers multiplied will be equal to those numbers added in non-encrypted form $D(E(x)E(y)) = x + y$ where E is encryption function and D is decryption function. This is called Additive Homomorphic Encryption. If one encrypted number is raised to the power of another number then that equals to the product of those numbers in decrypted form i.e., $D(E(x)^y) = xy$. Homomorphic Encryption can be performed using shared symmetric key i.e., same key is used for encryption and decryption, among participants without server being made aware of it. However, an evident fail case is when server joins network as a participant. It is worth mentioning that recent works in the domain try to tackle this problem by various means.

To summarize, SMC techniques may use secret sharing to generate keys for homomorphic encryption. Every participant encrypts their gradient and they all are combined by aggregator and once combined, atleast t out of n users are required collectively to decrypt the aggregated weights. Ideally t should be greater than $n/2$. Since individual participant gradients cannot be made known by the aggregator or other participants, this allows for Secure Aggregation. Furthermore, keys may be generated again after k number of rounds to make system more robust. Another approach for Secure Aggregation is not to just encrypt gradients but rather encrypt the complete model and perform training on it using encrypted data, however, activation functions like $ReLU$ and $tanh$ don't work while using Homomorphic Encryption so their polynomial approximations may be used in such defense.

Secure Aggregation may be used in combination with Differential Privacy and/or compression techniques. SMC is a solid defense but incurs a significant computation overhead. Moreover, it may fail against certain attacks.

4.2 DP: Differential Privacy

Differential Privacy is hiding information from an observer such that observer by looking at the output cannot determine if particular individual's data was used in the computation.
Differential Privacy can be implemented in FL using *Gaussian or Laplacian noise* to a participant's information i.e., gradient updates. DP can be used by clients as a defense against model inference attack or gradient leakage. Aggregator/server may also use DP to add noise after aggregation so an observer cannot guess which clients were participating in the training [22].
Differential Privacy is a robust defense that works in almost every scenario without any computational overhead but veracity of the model is affected by it.

4.3 TEE: Trusted Execution Environment

TEE allows users to defined private segments of memory that cannot be accessed or read by any other processes outside that segment. The code executing over that segment cannot be inspected by any process including operating system or hyper-visor on which that is running. This can help in dividing the program into trusted and untrusted parts. This can be used in Federated Learning for combining results without any other entity on the same machine or other machines having access to it.
One way of using TEE in Federated Learning is mixing layers of different clients randomly before aggregation [23].

4.4 Other Defenses

Compressing the model (using sparsity of information in it) can help reduce communication costs and provides data confidentiality [24]. Add Differential Privacy on each layer of model separately to prevent advanced inference attacks like GAN based attacks [18].

4.5 Server-side Defenses

Aggregator/server can perform various aggregation time defenses against adversarial participants such as, finding distance between global weights and local updated weights by participants and selecting only ones which are close to global model. In server cleaning, proxy server data (a generalized data hosted by server) can be used for filtering updates that deviate. Pruning can be done to filter updates from clients to prevent against attacks like model-poisoning. Using dimensionaity reduction techniques like Single Value Decomposition (SVD) to catch any outlier update and ignore those. Adversarial training i.e., training model to predict wrong, can be avoided by model sharing i.e. few powerful participants with large amounts of data share baseline models that other clients can follow, this will prevent against any training attacks. Defenses have been devised against formidable attacks like Sybil i.e., in which multiple participants perform attack on the system in collaborative way [18]. GANs, deep-learning-based generative model, can be used as Defense against adversarial gradients by classifiying updates into adversarial and benign. GANs can be used to add noise to updates to protect from inference attacks as well.

Chapter 5
Recommender Systems

Recommender Systems are the algorithms that suggest appropriate items to a specific user by building relationship between items and users based on some criteria e.g., recommending movies which may interest a particular user, based on the movies $m \in M$ that user already interacted with. Recommender systems research is characterized by a common problem area rather than a common technology or approach. Its aim is to lower the user's search effort by listing those items of highest utility. A key feature of such a system is providing a personalized view of data. Adomavicius and Tuzhilin (2005) formulated the problem as, ”*let C be the set of all users and let S be the set of all possible items that can be recommended. Let u be a utility function that measures the usefulness of item s to user c, i.e., $u : C\ x\ S \to \mathbb{R}_+$, where $\mathbb{R}_+$ are non-negative real numbers in a certain range. Then, for each user $c \in C$, we want to choose such item $s' \in S$ which maximizes the utility. More formally, $c \in C,\ s'_c = arg \max_{s \in S} u(c, s)$.*” [25]

The data used in recommender systems may include opinions i.e., ratings, reviews given by user explicitly or inferred from user's behaviour i.e., how user interacted with item(s), user's demographics i.e., age, gender, locality. Knowledge source for recommender algorithms can be classified into social, individual, content. Social includes pool of users' opinions, behaviours, demographics which could be used to predict what a particular user from a certain group might be interested in. Individual uses user's prior interactions with items which may include opinion, behaviour and demographic to suggest newer items. Contrary to these, content based knowledge finds the items similarity with other items based on item features, domain and context.
The information that system uses to make predictions is named *query or context* and the entities that are recommended are named *items or documents*. Recommender system performs candidate generation i.e., the predictive model finds the most relevant subset of items against a respective query. If predictive model uses content based knowledge then such system is called **Content Filtering** e.g., if user A watches two 'cute cat' videos then system recommends more cute animals videos to user A. If a model uses social and individual knowledge base then such system is called **Collaborative Filtering** e.g., if user A is similar to user B and user B likes cat videos then recommend cat videos to user A too. Even if A haven't seen any cat videos before.

5.1 Content Filtering Methods

Content-based Filtering uses same feature space for both user and item. For simplicity, consider each pair item and feature (or user and feature) having a binary value 1 or 0 representing if a particular item has that feature or not respectively. For a user, having a particular feature would mean that that user is interested in that feature. Then based on items which have those features that user is interested in, items can be suggested to that user. Similarity between items and users may be found using dot product, Euclidean distance or normalized cosine similarity where k items with highest similarity are recommended to the user. This approach does not requires other users' data to suggest item to a particular user. Moreover, as model is capturing specific interests of a user, it can recommend items which no other user may be interested in. Content based filtering is not used widely because it requires explicit feature engineering for predictions, the quality of predictions will only be as good as the features picked. Furthermore, it has limited ability to expand on users' existing interests.

5.2 Collaborative Filtering Methods

In Collaborative Filtering (CF), there is no predefined feature space. CF uses implicit features that are more useful as they can map aspects which are difficult to profile in Content-based Filtering. Recommendations are performed based on user's previous interactions. Recommendation method in which ratings from similar users are used to predict new items for target user is called User-based CF [33]. Method which uses similarity between items to recommend new items to a target user is called Item-based CF. User-based and Item-based methods are named as *Neighborhood Methods*. Another approach is to map users and items on latent factors inferred from rating patterns. Such methods are called *Latent Factor Methods*.

Collaborative Filtering methods can include implicit observations of user behaviour as well. Which can help in providing more sophisticated and problem specific recommendations. Filtering through a logic can also be applied e.g. logic may be put in place to not recommend movies which user already watched.

CF methods use previous interactions' memory combined with Machine Learning models or other prediction techniques to find missing ratings [26]. Dimensionality reduction can be used to improve predicted rating results as known ratings are mostly sparsed [27]. Since CF methods use user's previous interactions for prediction, it lacks in providing good results for new items or users (Cold-Start).

5.3 Inductive vs Transductive Models

Inductive Learning builds a *predictive model* that learns the relationship between features and target variables. The model is trained on training data such that it can predict/label data points that were not exposed during training. After training, if new data points are encountered, they can be labeled correctly using the pre-trained model.

Transductive Learning: "Transduction is the inference from particular to particular" [28], For a classifier that has a set of labeled and unlabeled data points and it has to be classified for this particular example rather than a general classification for unseen examples, the learning is Transductive. In this case, instead of using only labeled data, the model can leverage the information contained in the unlabeled data during the training to have better predictions. The information can be some sort of inherent relation between those data points which will help to improve the accuracy of the model for that particular set of unlabeled data points. The drawback is apparent from its idea that it will not be able to give predictions on any unseen data point.

Model's learning methodology can affect its usability and portability. Having an inductive learning model can suggest items to new users and also suggest new items to users without retraining the model. If the database is huge then it is not optimal to retrain model after every new user or item is added, hence it is preferable to use inductive recommendation systems.
Below is a list of some major recommendation system algorithms categorized by learning methodology:

Transductive:

- PMF (Probabilistic Matrix Factorization)
- GCMC (Graph Convolutional Matrix Completion)

Inductive:

- IGMC (Inductive Matrix Completion Based on Graph Neural Networks)
- IDCF-MF (InDuctive Model-based Collaborative Filtering Approach)
- F-EAE (Deep Models of Interactions Across Sets)
- Pinsage-KDD (Graph Convolutional Neural Networks for Web-Scale Recommender Systems)

5.4 Recommender System Techniques

In general setting of recommendation system, we have a set of users which are related to items. Relation can be defined as a utility of a certain item for a specific client. This relation is only known between few users and items e.g. in a movie recommendation system, the known relations could be the movies a user watched (liked or disliked based on which its utility would be found) and unknown relations would be the movies which user have not watched yet. User-Item relation will not be known in most cases i.e., a user would not have watched most of the movies present in the database. Recommendation systems use the known relations to predict unknown relations between users and items.

There are two main techniques to find these unknown relations: *Factorization and Link Prediction* methods.

5.4.1 Factorization

Factorization use Latent Factor models. It borrows the concept of matrix decomposition from Linear Algebra and decomposes rating matrix into user and item latent spaces which are optimized to minimize loss on the rating matrix.
A simple collaborative filtering embedding model is demonstrated here. Consider an adjacency rating matrix $A : m*n$, where m = number of users and n = number of items. Embedding matrix $U : m*d$, m rows of users having d dimensional embedding and $V : n*d$, n rows of items with d dimensional embedding
The embeddings are found so the product U^TV is close to the adjacency matrix A. Observe that the (i,j)th item of $U^T.V$ is the dot product $<U_i, V_j>$ of the embeddings of user i and item j, which should be close to $A_{i,j}$. Hence, U and V are found such that the error between resultant matrix $U^T.V$ and A is minimum.

Note: CF only means to filter the relevant item for target user, while considering a similarity which exists between user-user, user-item-user and item-item etc, using the Rating Adjacency Matrix. So it is not necessary to find complete embedding space for a prediction.

5.4.2 Link Prediction

The matrix completion problem is addressed by resolving it into a link prediction in a bipartite user-item graph. The missing entry of i^{th} user and j^{th} item in the rating matrix R is reduced to the missing link between i and j. An Undirected graph $G = (N, E)$ is constructed from R: rating matrix, where $N = N_u \cup N_v$ are the users and items represented as nodes and $E = (u, v, r_{u,v})$ is the set of labelled edges between N_u and N_v. Graph Neural Networks are used to exploit this structure. These models are able to learn diverse interactions across nodes. Although the learned embeddings in MF are highly expressive in terms of representing the user preferences but the technique contains an implicit bias towards the already seen data. Making it unable to capture the aspects of utility related to novelty.
Most of the Prior research in GNN based link prediction methods revolves around learning nodel level GNNs such as GCMC. PinSage [29] is the slight variation of GraphSage GNN model which is similar to the approach of GCMC but also uses the side information (i.e feature vectors) to enhance the performance. On the other hand, IGMC advocates to learns a graph level GNN on local subgraphs, proposing an inductive model without using side information.

5.5 State of The Art Baseline Algorithms

Following is a brief summary of most commonly used state-of-the-art Recommender Systems:

TRANSDUCTIVE METHODS

5.5.1 Matrix Factorization (MF)

To perform recommendations using Matrix Factorization, the users and items are mapped to a common latent space having dimensions k, such that user-item interaction is dot product in that latent space. Let U represent users latent matrix where each row i represents a user u_i^T having latent embeddings of size k. Similarly, let V represent item latent matrix where each column j represents an item v_j having embedding of size k. Then for any user u_i^T and item v_i, the rating user u_i^T gives to item v_j can be found by taking dot product of vectors u_i^T with v_i i.e., $r_{ij} = u_i^T.v_i$.
To populate the the latent vectors of item and user, regularized squared error is minimized on known ratings.

$$\min_{u*,v*} \sum_{i,j \in K} (r_{ij} - u_i^T v_j)^2 + \lambda(||u_i||^2 + ||v_j||^2)$$

Here K is set of all (u, v) pairs in r for which rating is known. Regularization term is used to prevent overfitting, λ is regularization factor.

The objective function can be minimized using:

- *SGD: Stochastic gradient descent*, a generic method to minimize loss functions.
- *Weighted Alternating Least Squares (WALS)*, a specialized method for this particular objective function.
- The embedding space can also be found with *Neural Network* models.

5.5.2 Graph Convolutional Matrix Completion (GCMC)

GCMC uses the problem of link prediction on bipartite graphs as the technique to implement matrix completion. This provides greater structured external information, and combined with interaction data, it can be a solution to the cold start problem[30].
In a bipartite graph, users and items are represented as nodes and the labeled edges between them denote observed ratings. We predict ratings by predicting labeled links in the graph.
The implementation is done trough a graph autoencoder based on differentiable message passing on

the bipartite graph. User and item embeddings the build through message passing on graph by the graph convolution layer of the encoder and using a decoder, new ratings are predicted. These new ratings the predictions of labeled edges.
GCMC is *transductive* because the entire ratings dataset is required to build the user-item bipartite graph. For a new user whose embedding is not available, the graph needs to be rebuilt.

INDUCTIVE METHODS

5.5.3 Inductive Matrix Completion Based on Graph Neural Networks (IGMC)

IGMC [31] is an inductive model which uses link prediction based interpretation of recommendation problem. Proposed algorithm is based on the idea that the local graph structure around the target user and target item are the effective predictor for the target rating. The model learns a graph level neural network on the locally extracted sub-graph around the respective target user and item. Enclosing subgraph extraction method can be referred below in Algorithm 1. The algorithm constructs h-hop subgraph around the user-item pair (u, v) where the neighbouring nodes are all the nodes at the 1-hop distance from the user and item nodes in U and V at any h-hop. These local enclosing subgraphs are fed to the graph neural network which learns a mapping from each subgraph to the target user and item. Learning this mapping, makes it independent of using any learned embedding or content side information for further predictions. The algorithm further claims that in node level GNN's such as pinSage and GCMC, the learned node embeddings are essentially independently rooted subtrees which fails to model the interactions captured by training of bi-rooted local enclosing subgraph.
R-GCN operator is used for computing the node feature at GNN layer *l*+1 as:

$$x_i^{l+1} = W_0^l.x_i^l + \sum_{r \in 0-k} \sum_{j \in N_r(i)} \frac{1}{|N_r(i)|} W_r^l.x_j^l$$

These node features $(x_i^0, x_i^1, x_i^2, ...x_i^l)$ are concatenated to form hidden representation h_i. A pooling layer then generates the final graph representation g by concatenating h_u and h_v. Final rating is predicted using basic MLP with the ReLu activation function and root mean squared loss for optimization.

Algorithm 1 ENCLOSING SUBGRAPH EXTRACTION

```
1:  input: $h$, target user-item pair $(u, v)$, the bipartite graph $G$
2:  output: the $h$-hop enclosing subgraph $G^h_{u,v}$ for $(u, v)$
3:  $U = U_{fringe} = \{u\}, V = V_{fringe} = \{v\}$
4:  for $i = 1, 2, \ldots, h$ do
5:      $U'_{fringe} = \{u_i : u_i \sim V_{fringe}\} \setminus U$
6:      $V'_{fringe} = \{v_i : v_i \sim U_{fringe}\} \setminus V$
7:      $U_{fringe} = U'_{fringe},\ V_{fringe} = V'_{fringe}$
8:      $U = U \cup U_{fringe},\ V = V \cup V_{fringe}$
9:      Let $G^h_{u,v}$ be the vertex-induced subgraph from $G$ using vertices $U, V$
10:     Remove edge $(u, v)$ from $G^h_{u,v}$.
11: end for
12: return $G^h_{u,v}$
```

Note: $\{u_i : u_i \sim V_{fringe}\}$ is the set of nodes that are adjacent to at least one node in V_{fringe} with any edge type.

5.5.4 InDuctive Model-based Collaborative Filtering Approach (IDCF-MF)

This algorithm implements an inductive collaborative filtering framework using matrix factorization [32]. IDCF is divided into two representation models. The first model using matrix factorization and factorizes a group of key users' rating matrix to get meta latents. The second model implements attention-based structure learning. This learns and predicts hidden relations and patterns between query and key users.

IDCF is inductive because of the meta latents that inductively compute embeddings for query users via neural message passing. The model enables inductive representation learning for users.

There are feature-driven models(as opposed to collaborative filtering models) that can be inductive but they don't have the same expressiveness and scalability because they are feature-dependent and high-quality features that can reveal user interests for personalized recommendation are hard to collect due to increasingly concerned privacy issues.

Chapter 6
Federated Recommender Systems

Using conventional machine learning methods for training recommendation algorithms requires users to send their data to the central server for training. The privacy preserving techniques in this setting are quite vulnerable and carries extensive assumptions for adversary, leaving the user exposed to never ending threat of privacy leakage. Moreover, the current GDPR laws restricts the mobility of users' private data from user devices. Most viable solution to this problem is to train the recommendation system in federated settings, as it facilitates the learning process in the said availability of data with high guarantees of privacy and fast convergence. Federated Learning in recommendation system is still quite a new research area where a lot of RecSys techniques are unexplored and most of the work done is centered around Matrix Factorization based algorithms. Furthermore, the federated implementation of RecSys algorithms does not completely ensure privacy as in basic Federated Learning. Hence there is a need to have a detailed analysis of the existing FedRecs in the framework of privacy and recommendation system.

6.1 Defining Federated Recommendation System

Federated recommendation system aims to collaboratively learn a recommendation model between multiple parties without their data being exposed to central server or any of the similar party they are collaborating with. FedRec can be categorized according to the data structure of RecSys in the perspective of three different federated settings (i.e Horizontal, vertical and Transfer). For simplicity, one can generalize the data structure to a matrix having users and items as rows and columns respectively. In horizontal FedRec, model will be trained on different parties mapping to the shared item set. In Vertical FedRec, the model has common rows (i.e users) whereas different columns or item set to map. Transfer FedRec is the merger of the above techniques where the model is being trained on the different parties mapping to different item set. Horizontal FedRec is the most relevant as it applies to the cases where the different parties are multiple users and the model is trying to learn there behaviour in the specific domain of already defined item set. Our focus will also be limited to the analysis in the horizontal setting.

6.2 Existing Algorithms

The following disscussion is focused on the Federated Recommendation Algorithms using MF and GNN based techniques. The papers omitted from the discussion are Meta-learning based Matrix factorization methods such as [33] and [34]. Others are slight variations of already mentioned algorithms.

6.2.1 Federated Collaborative filtering

Being a pioneer in Federated RecSys FCF, [35] proposed a federated implementation of Matrix factorization (MF), in which it is claimed that the algorithm does not necessitate the rating matrix to be stored on one device completely. The training step can be reduced to the single user where the respective user only has its own user embedding (i.e one corresponding row of user embedding matrix) and complete item embedding matrix which is shared across all users. In this training step, the particular user only needs to have its own set of ratings (i.e one corresponding row of rating matrix R). Hence, FCF can be defined in three integral parts: 1)Model parameters 2)Item Embedding Matrix and 3)User Embedding, where User embedding is private to every user while others are shared. FCF algorithm is as followed:

1) item embeddings Y are initialized and distributed by the server to every user U_i

2) predictive rating p of i^{th} user for j^{th} item is computed locally as $p_{(i,j)} = u_i^T.y_j$ (the step is repeated for all items in parallel on every user device)

3) loss for user embedding and item embedding is computed. User embedding vector is updated and the local update of item embedding is sent back to the central server.

4) Federated Averaging: server computes the weighted average of $\sum_{i=0}^{n} w_i * \delta Y_i$, where n is the number of users involved and w_i is the wight given to i^{th} user

5) The updated item embedding is sent back to users and the whole process repeats itself.

Algorithm 2 shows implementation of user-level federated collaborative filtering.

Algorithm 3 *Federated Collaborative Filtering(FCF).* n: users, m: items, $r_{i,j}$: rating by user i on item j, U: user profile matrix, V: item profile matrix, $< u_i, v_j >$: predicted $r_{i,j}$

```
Input: M Clients
Server
Initialize: Y
for each client m in M, in parallel do
    $\nabla Y^{(m)}$ = ClientUpdate($Y$)
    $Y = y - \eta \sum_m \nabla Y^(m)$
end for
function CLIENTUPDATE(Y)
    Update user factor $X_u$
    Compute item factor $Y^{(m)}, gradients \nabla Y^{(m)}$
    Return $\nabla Y^{(m)}$ to server
end function
```

6.2.2 Limitations of Vanilla FedMF

FCF [35], the vanilla implementation of federated matrix factorization, carries a number of issues in terms of privacy and recommendation problem. The privacy principle works on the assumption that by keeping the user embedding and corresponding row in the rating matrix locally, the data cannot be breached. As the aggregation step at the server's end is agnostic to the real representation user data. Furthermore, the nature of transductive inference in this implementation also limits its performance in federated setting. The detail of above stated limitations is as followed:

Value Leakage: For any basic model inversion attack, it is a trivial problem to reconstruct the user embedding vector from the updated item embedding and any other model parameters used. As discussed earlier in the chapter 3, the stated algorithms can regenerate the image used during the training. The algorithm works for batch sizes 1-8 quite well. In our case, that image representation can be replaced for the user embedding vector and the complexity reduces to batch size 1. Although the performance can be questioned with less number of local epochs, but for a federated environment where the model is targeted to reduce communication rounds, the cost is too much. The is addressed by using different defenses techniques such as Homomorphic encryption, secure multi-party computation, etc. The Defenses will be further elaborated in the discussion of respective papers.

Existence Leakage: With all the defenses to model inversion attacks, the privacy in item embedding still remains an issue in case of explicit feedback. The updated item embedding matrix of user i will have $\delta y_j > 0$ for only those values of j which were interacted by user i. The problem cannot be eliminated by using different encryption techniques. Obfuscation-based methods are used which add noise (eg: differential privacy). Although these algorithms provide good privacy guarantees at higher level of noising, it comes at the cost of compromising on the accuracy of the model.

Transductive inference: In a federated setting, such a model becomes impractical. The selected pool of edge devices for FL model training cannot correspond to the full rating matrix available in a non-federated setting. Furthermore, there is continuous updating of rating matrix values through edge devices. To have targeted results, the model needs to be re-trained periodically to incorporate all the changes and the lacking that is coming from having an incomplete Matrix. This computational cost is not affordable in a federated setting. Moreover, there is also a compromise in the accuracy of the global model due to the unavailability of a complete matrix in FL settings. The whole matrix is completed in multiple FL pools. For every pool, the embeddings will be optimized for that particular set of users. Being Transductive, the models cannot be independent of user x that is chosen in the selection. Every user has a direct impact on the learned embeddings of other users. The same property (i.e transductive inference) which helps to improve accuracy in non-federated setting is affecting the performance in FL.

6.2.3 Secure Federated Matrix Factorization

The Algorithm [36] resolves the problem of *value leakage* from model inversion attacks. Homomorphic encryption is used to encrypt gradients coming from the user side. The idea is to avoid using obfuscation-based methods and use an encryption technique which do not rely on third party server. The only assumption taken is that the users involved in the training are honest.

Algorithm 4 *Federated Matrix Factorization.* n: users, m: items, $r_{i,j}$: rating by user i on item j, U: user profile matrix, V: item profile matrix, $< u_i, v_j >$: predicted $r_{i,j}$

```
**init:** Server initializes V
**init:** User initializes U
**Output:** Converged U and V
Server has latest V which all users download
**User's local update**
Download V from server, perform local updates:
$u_i^t = u_i^{t-1} - \eta \nabla u_i F(U^{t-1}, V^{t-1})$
$gradient_i = \eta \nabla v_i F(U^{t-1}, V^{t-1})$
send $gradient_i$ to server
**Server Update**
Receive $gradient_i$ from user-i
Perform update: $v_i^t = v_i^{t+1} - gradient_i$
**WHERE**
$u_i^t = u_i^{t-1} - \eta \nabla u_i F(U^{t-1}, V^{t-1})$
$v_i^t = v_i^{t-1} - \eta \nabla v_i F(U^{t-1}, V^{t-1})$
$\nabla u_i F(U, V) = -2 \sum_{j:(i,j)} v_j (r_{i,j} - < u_i, v_j >) + 2\lambda u_i$
$\nabla v_i F(U, V) = -2 \sum_{j:(i,j)} u_j (r_{i,j} - < u_i, v_j >) + 2\lambda v_j$
```

Algorithm 3 in section 6.2.1 shows how FedMF is implemented. HE can be introduced in this algorithm to overcome the problems described earlier. Secure FedMF [36] uses the additive and distributive property of **Homomorphic encryption(HE)**. The averaging step done on the server side is on encrypted gradients. The encrypted gradients cannot be used in model inversion attack to regenerated user embedding. The decryption is done only on user side. Keeping the assumption that every user is honest, the secret key is shared among the users, making model independent of any third party server. The algorithm is as followed

Figure 6.1 shows an illustration of HE-based FedMF. This is divided into 3 parts.

1) **Key generation**: it is carried out by one of the users. Public key known to all users and server. Secret key is shared only between users. SSL channels are used to send secret keys between users.

2) **Parameter initialization**: item profile matrix V is initialized at server and user profile matrix U is initialized by each user locally.

3) **Matrix factorization**: Server first encrypts V using public key and gets ciphertext C_V. Each user then downloads latest C_V and decrypts it using secret key and gets the plaintext V. V is then used to perform local update and computation of gradient G. G is encrypted using public key again getting C_V. This is sent back to the server via SSL. Server updates V such that $C_V^{t+1} = C_V^t - C_G$. This step

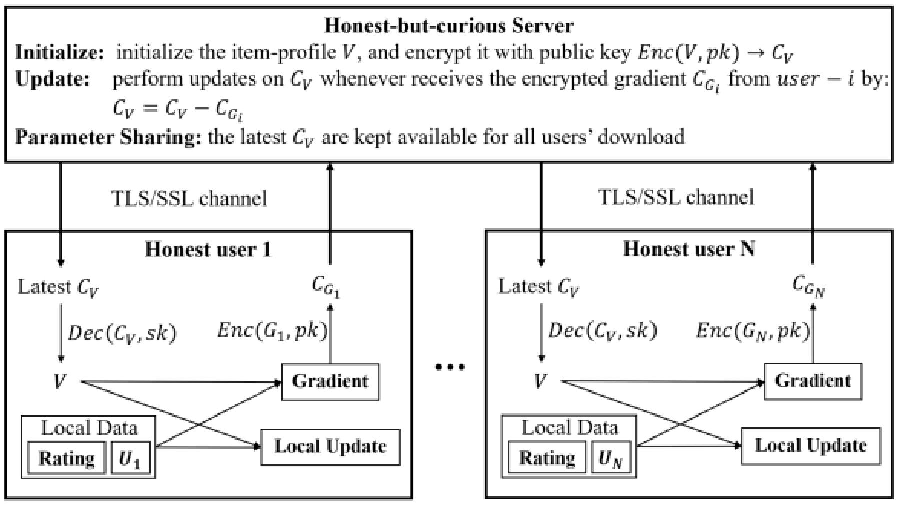

Honest-but-curious Server
Initialize: initialize the item-profile V, and encrypt it with public key $Enc(V, pk) \rightarrow C_V$
Update: perform updates on C_V whenever receives the encrypted gradient C_{G_i} from $user - i$ by: $C_V = C_V - C_{G_i}$
Parameter Sharing: the latest C_V are kept available for all users' download
TLS/SSL channel
TLS/SSL channel
Honest user 1
Latest C_V
C_{G_1}
$Dec(C_V, sk)$
$Enc(G_1, pk)$
V
Gradient
Local Data
Rating
U_1
Local Update
Honest user N
Latest C_V
C_{G_N}
$Dec(C_V, sk)$
$Enc(G_N, pk)$
V
Gradient
Local Data
Rating
U_N
Local Update

Figure 6.1: Secure FedMF Illustration. Reprinted from Secure Federated Matrix Factorization, IEEE, (2020). [36]

is done until convergence.

Limitations: Apart from having assumption of honest users, the model is also vulnerable to privacy leakage from item embedding. The δC_v would be non-zero for interacted items. As the mapping of item embedding is public, it is a trivial task to know the interacted items of the target user for any malicious party. With reference to recommendation algorithms, the approach inherits all the issues of Federated MF based implementations.

6.2.4 Fedrec: Federated Recommendation with explicit feedback

The paper [37] argues that the FCF [35] is vulnerable to the *existence leakage* in case of explicit feedback. The privacy by design only holds for the implicit feedback where $y_i \in (0, 1)$. The update:

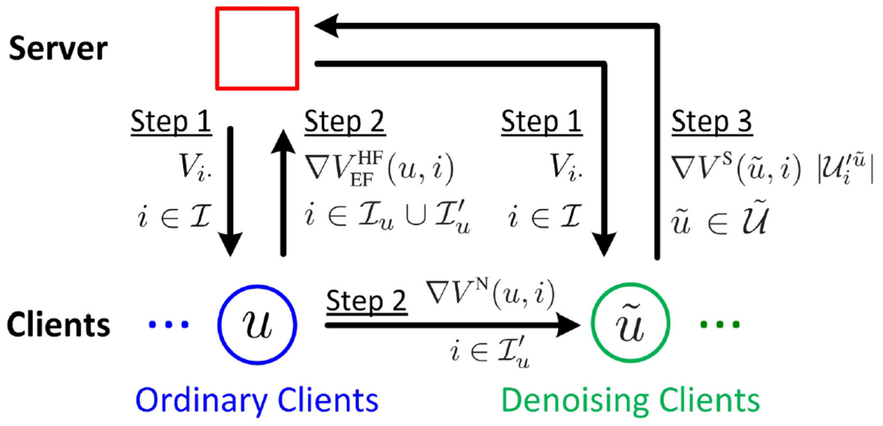
$$\nabla V_i(u,i) = (1 + \alpha y_{ui}).(U_u.V_i^T - y_{ui}).U_u$$

gives the negative feedback for non-interacted items keeping the $\nabla V_i(u,i)$ robust to leakage in case of implicit feedback. However same technique cannot be applied to the explicit feedback where it is not a binary class classification problem as the unrated items will make model biased towards the value chosen for the unrated item. The algorithm is subjected to omit the unrated items during the training, making the ∇V vulnerable to leakage.

FedRec [37] is an obfuscation based method which uses User averaging and hybrid filling strategy to randomly sample the non-interacted items $I'_u \subset \mathrm{I} \backslash \mathrm{I}_u$ and gives them virtual rating that is the computed average of $r_{ui} \forall i \in I_u$. This average scoring will not bias the learning process much and simultaneously achieving privacy goal. The sampling of unrated items will reduce computation time in comparison with FCF.Only limitation to the algorithm is its loss in accuracy, the user has to optimize between privacy goal and accuracy.

6.2.5 Fedrec++: Lossless Fedrec

The algorithm [38] proposes privacy-aware distributed denoising strategy for Fedrec [37], this strategy can be extended to other obfuscation based methods that are using pseudo-interacted items for preventing leakage from item embedding in explicit feedback.



Server
Step 1
$V_{i\cdot}$
$i \in \mathcal{I}$
Step 2
$\nabla V_{\mathrm{EF}}^{\mathrm{HF}}(u,i)$
$i \in \mathcal{I}_u \cup \mathcal{I}'_u$
Step 1
$V_{i\cdot}$
$i \in \mathcal{I}$
Step 3
$\nabla V^{\mathrm{S}}(\tilde{u},i)\ |\mathcal{U}_i'^{\tilde{u}}|$
$\tilde{u} \in \tilde{\mathcal{U}}$
Clients
u
Step 2 $\nabla V^{\mathrm{N}}(u,i)$
$i \in \mathcal{I}'_u$
$\tilde{u}$
Ordinary Clients
Denoising Clients

In Step 1, server initializes item embeddings $V_i : i \in I$ and broadcast it to all users. For step 2, all the ordinary users performs the training step on both real and pseudo ratings. Server will receive $\nabla V^{HF}(u,i)$ (i.e hybrid filtering) where $i \in I_u \cup I'_u$ whereas $\nabla V^n(u,i) : i \in I'_u$ is transmitted to denoising clients u" simultaneously. These clients then compute their updated embedding as followed:

$$\nabla V^s(u", i) = \textstyle\sum_{u \to u"} \nabla V^n(u,i) - \nabla V^{HF}(u", i)$$

Noise is then removed at the server end when aggregates the $\nabla V^s(u", i)$ from all the denoisers using equation:

$$\nabla V_i = \nabla V_i - \textstyle\sum_{u" \to U"} \nabla V^s(u", i)$$

This strategy solves the major limitation of obfuscation-based methods as there privacy guarantee was limited by the loss in accuracy caused by it. By increasing the number of pseudo items in Fedrec [37] one can achieve higher privacy goals against both *existence leakage* and *value leakage* problems. Distributed denoising strategy is also robust to privacy leakage as denoising clients do not know the identity of the user and only has ∇V for pseudo items. In case of server collaborating with these clients or any sort of man in middle attack the privacy breach is no more a trivial task.

6.2.6 Federated neural collaborative filtering

FedNCF [39] is the federated implementation of neural network based Matrix Factorization Methods. Neural MF is motivated by the idea that the simple inner product of item and user latent vector is inefficient in formulating the diverse relation between user-user, user-item and item-item. As these relations are learned implicitly in Matrix factorization based methods there has to be model which can learn these complex structures making use of non-linearity of neural network.
NuCF [40] is the fusion of the generalized MF (GMF) and MLP models. Both were proposed by the same author. GMF uses embedding layer to obtain latent user-item vector. These vectors are then fed to the linear layer which predicts the output by minimizing the Binary cross entropy loss. MLP concatenates user and item vectors into one single vector and the output is then fed to the hidden layer. NuCF fuses the output of both models and feeds them to another hidden layer after concatenating, where predictions are made. FedNuCF implements the model similar to the tradition FedMF. Model is seperated between public and private phases of learning. The item embedding and other model parameters are updated locally and are aggregated on the server. The implementation is for the implict feedback, privacy guarantees that are given are also subjected to it. For explicit feedback, there will be some modification in the NuCF model and a major change will be required Federated part for ensuring privacy preserving recommendation.

6.2.7 Federated Graph Neural Network

FedGNN [41] introduces federated implementation of link prediction based recommendation systems. These techniques use Graph neural networks to learn the higher order user-item interactions. In federated settings, the local user graph is not sufficient to train an accurate GNN model as the sparsity level is very high and the graph only contains the first order interactions. Hence, it is difficult to completely separate the local and public phases of model training. This difficulty requires the model to have privacy preserving model-update and graph expansion method.
FedGNN Framework: On each user device, local sub-graph is constructed from user item interaction and the neighbouring users requested from third party server (will be elaborated further). The embeddings for the respective entities are generated and fed to the graph neural network. Various kind of GNN can be used such as graph convolution network GCN [42], gated graph neural network GGNN [43] etc. The GNN model outputs the hidden representations of of item and user nodes. rating predictor uses these representations for further computing loss or just predicting in case of model being in a phase of validation. The server applies federated averaging on model gradients for GNN, rating predictor and user, item embeddings. Process repeats itself until convergence.
Privacy is ensured by two methods, one for protecting gradient leakage from the local updated model and other is the privacy preserving graph expansion method which ensures that the user expands its

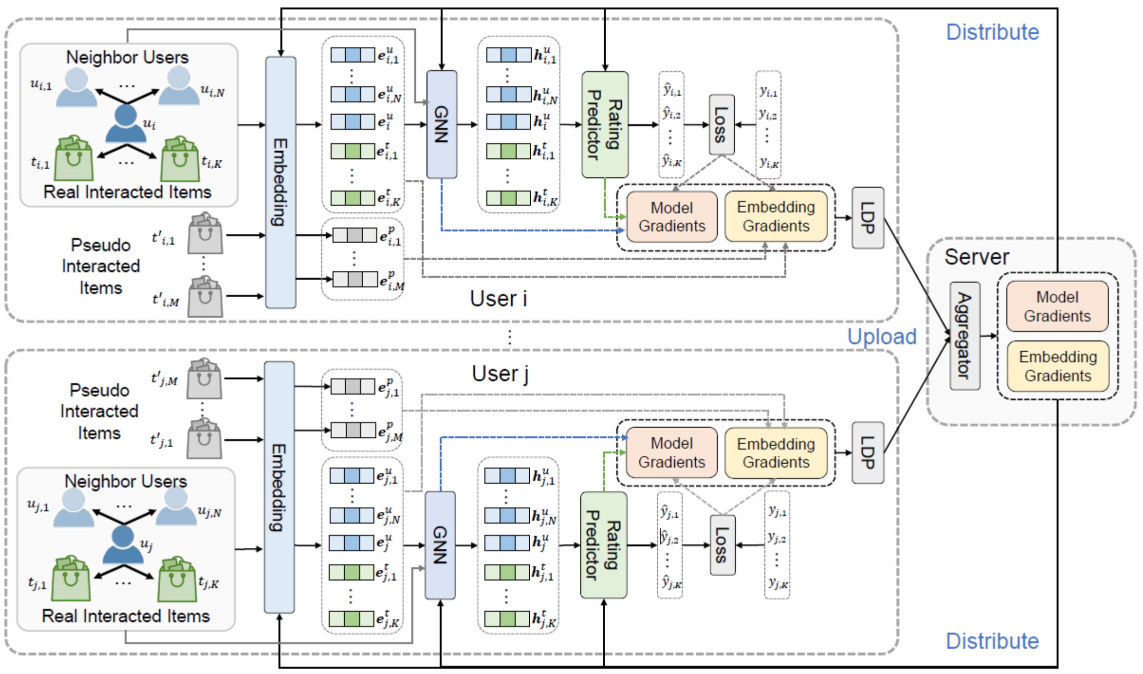


Figure 6.2: Framework for FedGNN approach. Reprinted from the work of Wu Chuhan (2021)

local graph without knowing the identity of its neighbour. The model update is highly prone to the gradient leakage as user only has the non-zero item embeddings for the items interacted. Furthermore, the graph structure sent for aggregation is more detailed than the item embedding in case of FedMF. The amount of local training is also more than the FedMF, making it easier for the model inversion attack to regenerate the local representation of data. The solution proposed is to use dual defense of pseudo interacted items and local differential privacy technique to address the leakage from non-zero item embedding and gradient leakage respectively. For graph expansion, a third party trusted server collects the all encrypted interacted item ids for every user. Using item matching, server distributes neighbouring users on the basis of common interacted items. User decrypts the embeddings on receiving and expands its local sub-graph. This way the central server do not receives the local data of user. The third party server is also agnostic of the item id's received as they are encrypted.

Limitations: There are mainly two limitations of this approach

- The privacy graph expansion method is preserved on the assumption that that third party server is honest but curious, that it will not try to access the encryption keys to uncover item identities.
- The major limitation is that this model is **transductive**. The rating predictor uses learned user and item embeddings, for any user not involved in the training will not be able to give predictions as its user embeddings won't be known.

6.3 Conclusion

The recommendation systems relying on the learned embeddings can pose a privacy challenge in FL. Most of the work we discussed revolves around the defenses for privacy leakage hence some privacy guarantees can be assured in current FedRecsys. However, no work has been done to address the transductive nature of these algorithms.

Chapter 7
Our Work

There is no inductive solution for federated recommender system problem as of yet. So this paper tries to deal with this problem. There could be two different approaches for this, one is to make a federated recommender system inductive e.g. making FedGNN inductive. Second is to make inductive recommender system federated e.g. making IGMC or IDCF federated. We propose solutions with each approach. First we propose & implement inductive version of FedGNN, **I-FedGNN**. Then we propose & implement federated version of IDCF, **Fed-IDCF**.

7.1 FedMF Baseline

To get started, we implemented vanilla FedMF. FLMF [44] is a github repository of our baseline implementation, can be referred to for further enquiry. We have simulated the environment for user updates, the training step for every single user is modulated where it runs the model only in its stated domain. Our main hypothesis was based on the idea that transductive models will perform badly in federated settings. As in FL environment, one cannot guarantee about the set of users to be involved in the training for every training round. The users involved in the training changes. In federated setting, We have analyzed our baseline model FLMF [44] for varying participants in every round. the dataset used was ml-latest-small. The participants were randomly sampled in 4 experiments with (30-100%) participants. the accuracy was seen to be continuously decreasing as the intersecting participants for every round reduces. The **Figure 7.1(a)** shows the obtained test curves for the said setting. **Figure 7.1(b)** is the scaled version of same graph to differentiate between the curves on point of convergence. Total number of rounds selected were 30 and 1 local epoch. Another hypothesis that the model performs adversely when the local epochs are increased. The stated hypothesis is tested in **Figures 7.2 7.7**, which shows that with increasing number of epochs the difference between test-loss and train-loss curves increases and the overall accuracy and convergence is also compromised.

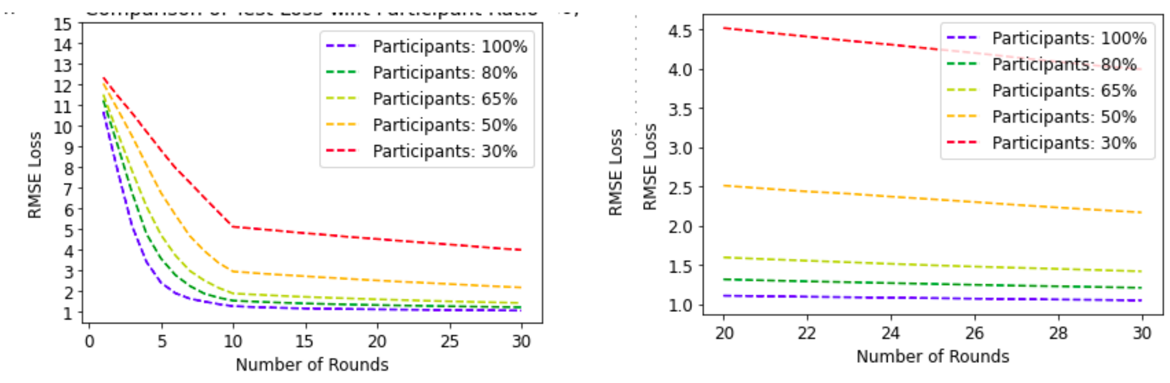


Figure 7.1: FedMF - Comparison of convergence with varying Participants

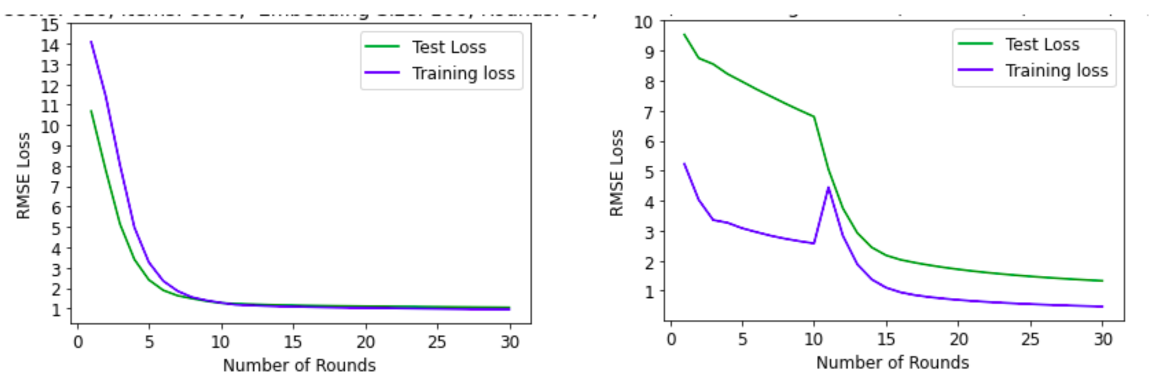


Figure 7.2: FedMF - Varying local epochs 01-05

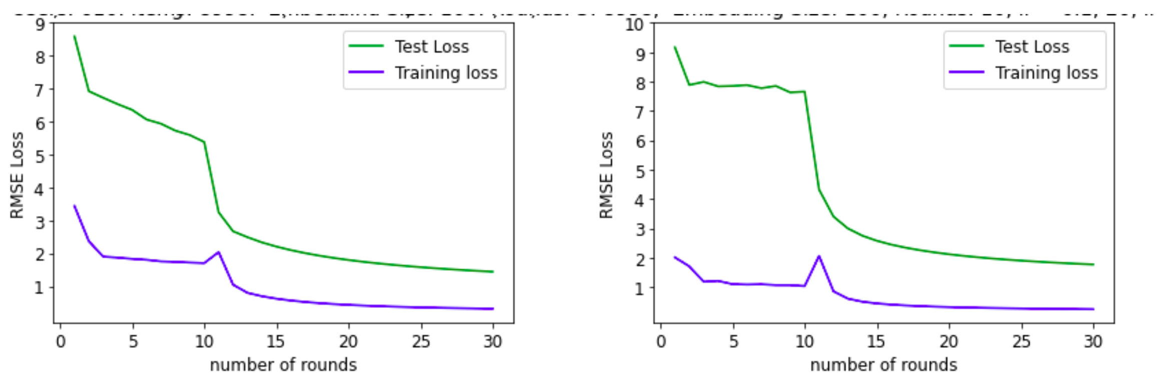


Figure 7.3: FedMF - Varying local epochs 10-20

7.2 Federated & Inductive GNN (FedIGNN)

As previously discussed, FedGNN has two major drawbacks, one is it being trasnductive and other one is potential security risk which can occur due to malicious third party. Second drawback is not an immediate threat and most privacy preserving schemes do employ such methods, so it is a reasonable assumption that third party is honest but curious.

The only major problem now is that FedGNN is not inductive, i.e. network has to be retrained to incorporate untrained (new) users, since new users don't have their user embeddings defined. We propose a solution for tackling this problem locally without retraining the entire network.

7.2.1 Problem Formulation

Consider a set of users U that have interactions with set of items I where a specific user u has interactions with a subset of items $i_1, i_2, ...i_k$ from I. This forms a bipartite graph between users and items.

Data is present on isolated user devices, so each user has a local first-order sub-graph based on observe ratings where user node is connected to items it has rated. Now using these subgraphs, the goal is to train a model which can predict rating for unobserved items for any user.

7.2.2 Framework

The system consists of central aggregator or server with a pool of clients. Server chooses 128 clients each round from these pool of clients for training. Initially, server sends an untrained model to all users for training. Then at each round, users train on that model and communicate gradients to the server. Server aggregates these gradients using FedAvg and sends the updated model to all users. Then the next training round begins and same procedure is carried on and on until convergence is reached.

On client side, the training round begins by converting subgraph nodes into their corresponding node embeddings (user and item embeddings). These embeddings are feeded into a GNN (GAT in this case) and hidden representations are obtained. Then the hidden representation vector of user embedding is taken dot product with each item's hidden representation vector to get ratings vector on which loss is computed using RMSE loss function. The learnable parameters are the GNN weights, the item and user embeddings. While item and GNN weights' gradients are communicated with server after training, the user embedding of each user is maintained locally for privacy preservance and not shared with server.

For improving results, neighboring users' embeddings are also used in local training, this is done through a third party protocol which ensures privacy (as discussed earlier in this paper). Secondly, for privacy reasons, differential privacy is employed along with pseudo-interacted items which prevents existence leakage.

7.2.3 Implementation

Since the code of this paper was not available online, we have implemented it from scratch. The original paper used following setup hyperparameters:

- GAT as GNN
- SGD as Optimizer with Minibatches
- Learning Rate 0.01
- Local Epochs Count: 3
- Clients per round 128
- Embedding size 256
- Gradient clipping threshold 0.1
- RMSE as Loss Function
- Epoch threshold for including neighboring user's embeddings: 2
- Pseudo Interacted Items Count: 1000
- Laplacian noise with LDP module 0.2 for Differential Privacy

Paper however, was unclear about how the embeddings should be initialized, so in our implementation, we have use xavier uniform for random initialization of the embeddings. We also used **Adam** as optimizer because SGD was not producing good results. We have skipped minibatches, and security features for now like pseudo interacted items and LDP.
For GAT specification, we have tried GATs with more than layer, but results were not satisfying, so we used GAT with one layer. As for the number of heads, GAT works reasonable with one head and training gets little noiser with more heads so we used a single head as well.

7.2.4 FedGNN Evaluation and Experiments

For evaluating our implemented model we have used 100k MovieLens data. Original FedGNN paper has RMSE of 0.92 on same data. Here we evaluate performance of our implementation. First we evaluate it using the SGD as optimizer. But as seen in the figure not much of a training can be

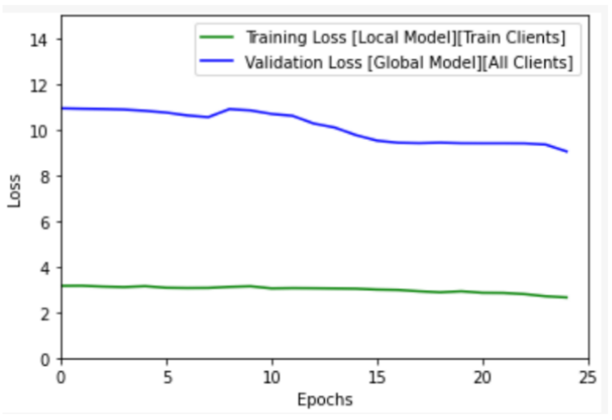


Figure 7.4: Fed-GNN using SGD as otpimizer

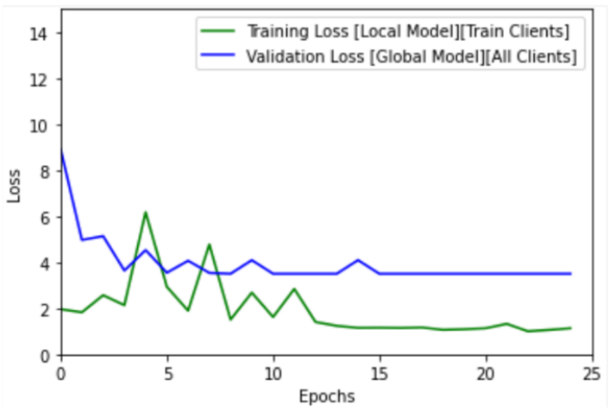


Figure 7.5: Adam with 2 GAT layers

performed using SGD, so we then use Adam. First with 2 layers, then with single. As it can be seen, the results with Adam and single layer of GAT are the most suitable.

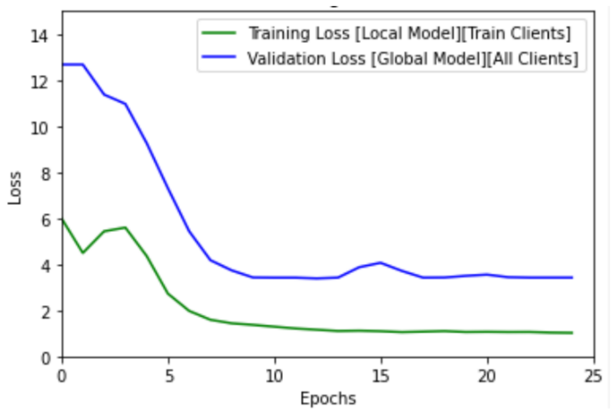


Figure 7.6: Adam with single GAT layer

7.2.5 FedIGNN: Inductive Learning

As users which have not participated in training will have undefined user embeddings, this would lead them to produce bad recommendations. To fix this, we usually need to retrain the network. But in this paper we propose tweak to locally train user embeddings to fix the problem of retraining the entire model everytime.

When a new user comes, we fix all the other item embeddings and model weights and only learn the user embeddings. This tweak can turn an transductive model into inductive.

Preliminary results have been shown below. This can be improved by performing training on randomly generated embeddings multiple times and choosing the best one. Or by increasing the overall embedding size so it can adjust better to the model. Code for this implementation can be found at: *https://github.com/beingbat/fedgnn-implementation.git*

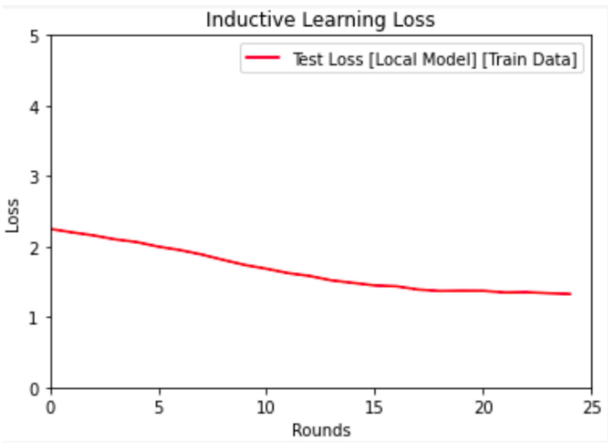


Figure 7.7: Learning User Embedding using test/train split of 0.3/0.7

7.3 Federated Inductive Model-based Collaborative Filtering Approach (FedIDCF)

There are two phases for this model, learning meta latents from a traditional transductive model and learning relational model between meta latents and new query users for inductive inference. For meta latents we have used our Federated Matrix Factorization implementation as a backbone and then proposed a federated implementation of the second phase of IDCF. We have also addressed the added challenge of privacy that came with it. The IDCF algorithm presents a hypothesis that there exist a unique linear combination of key user embeddings (i.e meta latents) for every query user which minimizes the loss for it. Mathematically, every query user embedding $\hat{p}_{\hat{u}}$ is a linear combination of key user embedding matrix P_k, which is $\hat{p}_{\hat{u}} = c_u^T P_k$, c_u^T is a vector containing weighted edge from query user $\hat{u}$ to every key user. In other words, user embedding of a query user (new user) is the weighted sum of key user embeddings.

7.3.1 Problem Formulation

A model has to be learned which can inductively compute the weighted edge ($c_{\hat{u}u}$) from query-key user ($\hat{u}$–u) using the embeddings of items that query user interacted with and the learned embedding of key user.

$$c_{\hat{u}u} = \frac{e^\top [W_q d_{\hat{u}} \oplus W_k p_u]}{\sum_{u_o \in U_k} e^\top [W_q d_{\hat{u}} \oplus W_k p_{u_o}]}$$

where $e^\top$, W_k and W_q are learnable parameters, $\oplus$ denotes the concatenation, $d_{\hat{u}} = \sum_{i \in I_{\hat{u}}} q_i$ where $I_{\hat{u}}$ includes the historically rated items of query user $\hat{u}$ and q_i is the item embedding of i^{th} item. These weights are used to compute the query user embedding $\hat{p}_{\hat{u}}$:

$$\hat{p}_{\hat{u}} = c_u^\top P_k$$

The embedding $\hat{p}_{\hat{u}}$ is then used to compute predicted rating:

$$\hat{r}_{\hat{u}u} = f_\theta(\hat{p}_{\hat{u}}, q_i)$$

The goal is to learn perform the above steps locally and learn the parameters $e^\top$, W_k and W_q in federated setting which will make the model capable of generating user embedding for new user at run time (i.e inferring inductively).

7.3.2 Implementation

For the backbone FedMF, we have used Secure Multi Party Communication for gradient communication as it is most robust among the existing algorithms. According to the two parameters, model inversion robustness and existence leakage immunity, the protocol has high privacy guarantees. For the training step to be performed in second phase, only thing that is not available locally is the key users embedding matrix P_k for which we proposed or privacy preserving pseudo mata-latent generator. The algorithm resolves the privacy leakage issue for key users as their embeddings will not be shared instead they are used only for computing the pseudo meta latents.

Algorithm 5 FedIDCF: Inductive Relational Model

```
 1: Server Side:
 2: Initialize: $W_q, W_k, e^\top$
 3: for r rounds: do
 4:     for each user $u$ in $U$ do
 5:         $W_q, W_k, e^\top = W_q, W_k, e^\top + ClientSide(W_q, W_k, e^\top)$
 6:     end for
 7:     FedAvg($W_q, W_k, e^\top$, n)                                ▷ n: number of users
 8: end for
 9: function CLIENTSIDE($W_q, W_k, e^\top$)
10:     $P_k = PseudoEmbediingMatrix()$
11:     $d_{\hat{u}} = \sum_{i \in I_{\hat{u}}} q_i$
12:     for n do local epochs:
13:         $c_{\hat{u}u} = \frac{e^\top [W_q d_{\hat{u}} \oplus W_k p_u]}{\sum_{u_o \in U_k} e^\top [W_q d_{\hat{u}} \oplus W_k p_{u_o}]}$
14:         $\hat{p}_{\hat{u}} = c_u^\top P_k$
15:         $\hat{r}_{\hat{u}} = f_\theta(\hat{p}_{\hat{u}}, q_i)$
16:         loss := $SSE(\hat{r}_{\hat{u}}, r)$
17:         $optimizer.step()$                                       ▷ Adam
18:     end for
19: return: $SMC(\nabla W_k, \nabla W_q, \nabla e)$                  ▷ Send gradients using SMC
20: end function
```

Pseudo Meta-Latents: Every key user encrypts their user embedding using homomorphic encrypption and sends the embedding to third party server. Server concatenates these vectors to have n dimensional encrypted key user embedding matrix. Using the additive and distributive property of homomorphic encryption server finds the similar n dimensional matrix which contains columns that

```
Minimum accuracy:  17.142857142857142
Maximum accuracy:  95.1219512195122
Mean of accuracy:  53.67203676007804
Median accuracy:  54.23728813559322
```

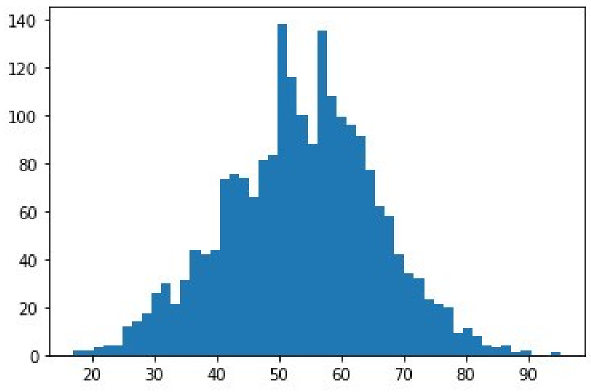


Figure 7.8: Accuracy distribution for query users

are linear combination of the actual matrix. This matrix of pseudo meta latents is decrypted when they are received by the clients. The key concept is that the new columns (pseudo meta latents) will span the same vector space as it was done by the real key user embeddings. These pseudo meta latents can not be mapped to real embeddings unless the third party server communicates the linear combination used to generate them to the malicious user.

Algorithm: On Receiving the pseudo-meta latents, every user decrypts them and initializes n-dimensional weight vector $c_{\hat{u}}$. We are not solving a cold star problem, hence we assume that there exist some items which query user $\hat{u}$ has rated through which $d_{\hat{u}}$ is computed. Weight with every pseudo key user is computed and divided by the sum of all weights to keep the combination convex. The parameters except $e^{\top}$, W_k and W_q are fixed. Predicted rating vector is computed and the loss SSE (sum of squared errors) is propagated backwards. On k epochs, the three parameters are sent to the server for the federated averaging step. For further insight, one can refer to 7.3.2.

7.3.3 Experiments

Experiments were performed on douban 100k dataset. number users and items were 3000 and ratings were above 100k. For local epoch 3, we have achieved the RMSE: 0.96047 which is close to the state of the art traditional transductive federated Recommendation System.

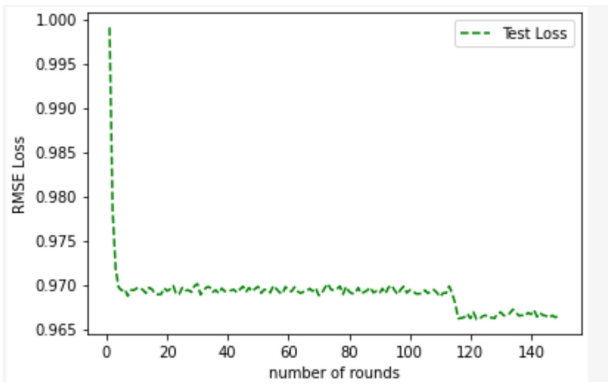


Figure 7.9: learning rate: 0.01

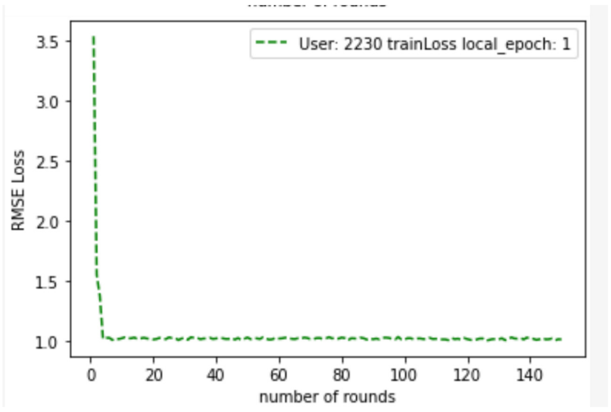


Figure 7.10: user id: 2230

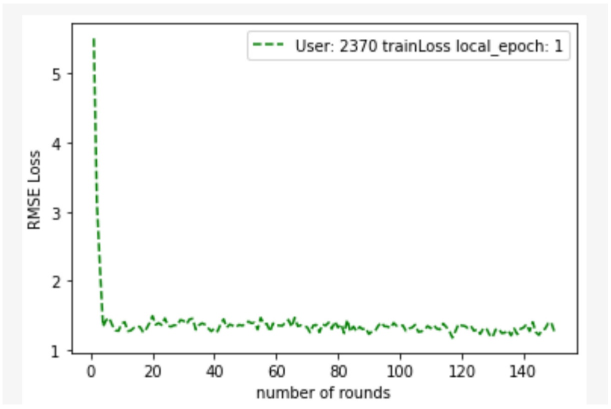


Figure 7.11: user id: 2370

Chapter 8
Discussion

8.1 Conclusion

We have proposed two algorithms to create an inductive and federated recsys, a combination which does not exist in the literature yet. These ideas however are in there early stage and can be improved upon greatly. Our major goal was to show that such implementation of recsys is possible which is inductive while being in federated setting which we have achieved in this paper. There are many schemes which have not discussed in this paper in detail that can be looked upon for creating more robust system e.g. deploying blockchain inspired mechanism for privacy etc.

8.2 Future Work

There are certain constraints with decentralized training e.g. weak clients are dropped, or noise is added or data distributions vary. All of these factor lead to model performing a little lower than they will do in centralized setting. This however, can be improved upon by using side information. Currently, federated recsys don't use side information, though using it can improve results if used together with collaborative learning.
In this paper we have discussed the security and transductive inference challenges in FL. However, there are two other important challenges as well, namely system and statistical heterogeneity. That can be further explored as well.
As for Recsys, an unexplored direction in FL environment is Auto-encoder based methods such as F-EAE [45] or VAE-SGD [46] are the state of art, where VAE-SGD is currently used in Netflix recommendations.

References

[1] J. K. H. B. M. F. X. Y. P. R. A. T. S. D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *NIPS Workshop on Private Multi-Party Machine Learning (2016)*, 2016.

[2] K. P. Murphy, “Machine learning: a probabilistic perspective,” 2012.

[3] T. C. Q. Yang, Y. Liu and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems*, 2019.

[4] A. T. T. Li, A. K. Sahu and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Processing Magazine*, 2020.

[5] D. R. S. H. B. A. y. A. H. B. McMahan, E. Moore, “Communication-efficient learning of deep networks from decentralized data.” 2017.

[6] B. K. A. M. H. B. M. S. P. D. R. A. S. Keith Bonawitz, Vladimir Ivanov and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” *Google*, 2017.

[7] M. Z. M. S. A. T. V. S. Tian Li, Anit Kumar Sahu, “Federated optimization in heterogeneous networks.” [Online]. Available: https://arxiv.org/abs/1812.06127

[8] M. M. S. J. R. S. U. S. A. T. S. Sai Praneeth Karimireddy, Satyen Kale, “Scaffold: Stochastic controlled averaging for federated learning.” [Online]. Available: arXiv:1910.06378v4

[9] Y. S. D. P. Y. K. Hongyi Wang, Mikhail Yurochkin, “Federated learning with matched averaging,” *ICLR*, 2020. [Online]. Available: arXiv:2002.06440

[10] S. M. S. V. K. N. Lau, “Model compression for communication efficient federated learning,” *IEEE*, 2021.

[11] J. W. C. Y. Y. Chen, X. Qin and W. Gao, “Fedhealth: A federated transfer learning framework for wearable healthcare,” *IEEE Intelligent Systems*, 2020.

[12] Kelvin, “Introduction to federated learning and challenges,” *Towards Data Science*, 2020.

[13] J. K. A. Y. D. N. Y. Liu, R. Zhao and J. Peng, “Towards communication-efficient and attack-resistant federated edge learning for industrial internet of things,” 2020. [Online]. Available: arXiv:2012.04436

[14] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, 2009.

[15] T. C. Q. Yang, Y. Liu and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2019.

[16] ——, “Learning from non-iid data,” *xzhu0027.gitbook.io*, 2020.

[17] V. S. Tao Yu, Eugene Bagdasaryan, “Salvaging federated learning by local adaptation,” 2022. [Online]. Available: arXiv:2002.04758

[18] A. U. Gorka Abad, Stjepan Picek, “Sok: On the security privacy in federated learning,” 2021. [Online]. Available: arXiv:2112.05423

[19] S. H. L. Zhu, Z. Liu, “Deep leakage from gradients,” *NeurIPS*, 2019.

[20] H. D. M. M. Jonas Geiping, Hartmut Bauermeister, “Inverting gradients - how easy is it to break privacy in federated learning?” 2020. [Online]. Available: arXiv:2003.14053

[21] R. S. A. S. S. I. S. N. P. Franziska Boenisch, Adam Dziedzic, “When the curious abandon honesty: Federated learning is not private,” 2021. [Online]. Available: arXiv:2112.02918

[22] M. D. C. M. H. H. Y. F. F. S. J. T. Q. S. Q. H. V. P. Kang Wei, Jun Li, “Federated learning with differential privacy: Algorithms and performance analysis,” 2019.

[23] C. F. Do Le Quoc, “Secfl: Confidential federated learning using tees,” 2021.

[24] B. G. Irem Ergun, Hasin Us Sami, “Sparsified secure aggregation for privacy-preserving federated learning,” 2021.

[25] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[26] D. H. John S. Breese and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” 2013.

[27] T. M. K. Xiaoyuan Su, “A survey of collaborative filtering techniques, advances in artificial intelligence archive,” 2009.

[28] A. Gammerman, V. Vovk, and V. Vapnik, “Learning by transduction,” *arXiv preprint arXiv:1301.7375*, 2013.

[29] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.

[30] M. W. Rianne van den Berg, Thomas N. Kipf, “Graph convolutional matrix completion,” 2017. [Online]. Available: arXiv:1706.02263

[31] M. Zhang and Y. Chen, “Inductive matrix completion based on graph neural networks,” *arXiv preprint arXiv:1904.12058*, 2019.

[32] X. G. J. Y. H. Z. Qitian Wu, Hengrui Zhang, “Towards open-world recommendation: An inductive model-based collaborative filtering approach,” *ICML 2021*, 2021. [Online]. Available: arXiv:2007.04833

[33] Y. Lin, P. Ren, Z. Chen, Z. Ren, D. Yu, J. Ma, M. d. Rijke, and X. Cheng, “Meta matrix factorization for federated rating predictions,” pp. 981–990, 2020.

[34] S. Lin, G. Yang, and J. Zhang, “A collaborative learning framework via federated meta-learning,” in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 289–299.

[35] M. Ammad-Ud-Din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, “Federated collaborative filtering for privacy-preserving personalized recommendation system,” *arXiv preprint arXiv:1901.09888*, 2019.

[36] D. Chai, L. Wang, K. Chen, and Q. Yang, “Secure federated matrix factorization,” *IEEE Intelligent Systems*, vol. 36, no. 5, pp. 11–20, 2020.

[37] G. Lin, F. Liang, W. Pan, and Z. Ming, “Fedrec: Federated recommendation with explicit feedback,” *IEEE Intelligent Systems*, vol. 36, no. 5, pp. 21–30, 2020.

[38] F. Liang, W. Pan, and Z. Ming, “Fedrec++: Lossless federated recommendation with explicit feedback,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 5, 2021, pp. 4224–4231.

[39] V. Perifanis and P. S. Efraimidis, “Federated neural collaborative filtering,” *Knowledge-Based Systems*, p. 108441, 2022.

[40] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[41] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, “Fedgnn: Federated graph neural network for privacy-preserving recommendation,” *arXiv preprint arXiv:2102.04925*, 2021.

[42] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[43] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[44] M. A. Rahman, *Federated Baseline Matrix Factorization*, ver. 2, Lahore, Punjab, Pak, 2022 [Online]. [Online]. Available: https://github.com/abdurahman02/FLMF-Federated-MF

[45] J. Hartford, D. Graham, K. Leyton-Brown, and S. Ravanbakhsh, "Deep models of interactions across sets," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1909–1918.

[46] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 689–698.