Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

PRIVACY-PRESERVING RECOMMENDATION SYSTEM USING FEDERATED LEARNING

by
Rahul Basu

Federated Learning is a form of distributed learning which leverages edge devices for training. It aims to preserve privacy by communicating users' learning parameters and gradient updates to the global server during the training while keeping the actual data on the users' devices. The training on global server is performed on these parameters instead of user data directly while fine tuning of the model can be done on client's devices locally. However, federated learning is not without its shortcomings and in this thesis, we present an overview of the learning paradigm and propose a new federated recommender system framework that utilizes homomorphic encryption. This results in a slight decrease in accuracy metrics but leads to greatly increased user-privacy. We also show that performing computations on encrypted gradients barely affects the recommendation performance while ensuring a more secure means of communicating user gradients to and from the global server.

PRIVACY-PRESERVING RECOMMENDATION SYSTEM USING FEDERATED LEARNING

by
Rahul Basu

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology and
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Data Science

Department of Computer Science
Ying Wu College of Computing

May 2020

APPROVAL PAGE

PRIVACY-PRESERVING RECOMMENDATION SYSTEM USING FEDERATED LEARNING

Rahul Basu

Guiling Wang, Thesis Advisor — Date
Professor, New Jersey Institute of Technology

Zhi Wei, Committee Member — Date
Professor, New Jersey Institute of Technology

Ioannis Koutis, Committee Member — Date
Associate Professor, New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Rahul Basu

Degree: Master of Science

Date: May 2020

Undergraduate Education:

- Master of Science in Data Science,
  New Jersey Institute of Technology, Newark, NJ, 2020
- Bachelor of Technology in Computer and Communications Engineering,
  Manipal Institute of Technology, Manipal, India, 2017

Major: Data Science

Presentations:

Rahul Basu, 'Reinforcement Learning and Creative Decisions in Video Games' *Workshop*, Manipal Institute of Technology, 2016

I dedicate this thesis to my parents who allowed their only son to remain halfway across the world through what has possibly been the most challenging time in our lives so I could finish my master's degree.

ACKNOWLEDGMENT

This thesis has really been a collaboration of the efforts of a lot of people and I owe a lot of it to them. I would firstly like to thank Professor Guiling Wang for believing in me by taking me under her tutelage and introducing me to Federated Learning. I would also like to thank her for checking in regarding my well-being in these uncertain times and guiding me in the right direction during the course of my thesis by offering invaluable advice, of which this thesis is a result. I would like to thank everyone at NJIT that I came into professional contact with starting with Professor Senjuti Basu Roy who I learnt a lot about research from and who gave me the opportunity to explore research during my first year at NJIT, Dong Wei who allowed me to learn from his experiences as a PhD student and has been someone I can look up to professionally, and Connor Watson who offered me valuable advice at a crucial time. I'd like to thank my co-workers and technical managers at Verizon 5G Labs in New York City with whom I was excited to discuss technology and some of whom told me about interesting findings in my field of research and outside of it; my friends Sattvik Sahai and Vishal Kuruganti of Stevens Institute of Technology and Projjol Banerjee and Pranay Pareek of New York University for inspiring and broadening my research horizons with their discussions. Most importantly, I would like to thank my parents for providing me with the financial support that I needed during the course of this thesis.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation for Federated Learning

Machine Learning is an alternate path to finding solution to problems which stands in stark contrast to the traditional philosophy of algorithmic programming, wherein rules and parameters are fed in and a result is presented as an output. In Machine Learning, results and parameters are fed in and the rules are deduced. This serves as the foundation for solving previously unsolvable problems such as handwriting recognition, analyzing large amounts of audio recordings consisting of high-dimensional data which is almost impossible to inspect manually. Machine Learning can help find the patterns instead which would help with interpreting audio signals [47].

For many years now, this new programming paradigm has been applied to a large number of fields and resulted in revolutionary technologies. It has been hailed as "the new electricity" and is set to change the world as we know it to the point where it has been theorized that future generations might look back and wonder how we lived without AI being an integral part of our everyday lives. Speech recognition, recommender systems, natural language processing and computer vision are based on data-driven learning systems. In fact machine learning based algorithms (usually deep reinforcement learners) have been able to beat the best of humanity at their own games [100]. Most of the state-of-the-art today is based on learning systems that requires data [66] [6].

Ideas and techniques that weren't as successful before and which resulted in the AI Winter [52] have re-emerged and turned out extremely successful. For instance, back-propagation, a concept from the 1970s [44] [72] serves as the backbone for Deep

Learning. There are many factors which contributed to this, of course. Usage of optimizers like Adam [60] have greatly reduced the amount of tuning necessary for Gradient Descent to work, for instance. Another reason is definitely the increase in computational resources freely available as well as the highest end computing power known to mankind. The processors in our smart wristwatches are more powerful than the processor that was used to first send human beings to the moon. Most importantly though, the explosion of data is what primarily contributed to this and allowed deep learning to unleash its true potential.

With more data, machine learning algorithms had a much larger sample space to decide what data was truly important and find patterns that were interesting. [45] makes a case for how having more data would make for better machine learning models. The predominant way of using machine learning nowadays involves collecting all this data in a data center. The model is then trained on powerful servers. However, this data collection process is often privacy-invasive. Many users do not want to share private data with companies, making it difficult to use machine learning in some situations, for instance when medical data is involved. Even when privacy is not a concern, having to collect data can be infeasible. For example, self-driving cars generate too much data to be able to send all of it to a server

Federated Learning is a newer paradigm in which the data is not collected and instead the parts that require access to the data are moved to the users' devices and never leave it. The model is collaboratively trained using locally available data and only the parameters are sent back to the server to make improvements to the global model. Federated learning approaches can lead to better models as a result because of the potentially larger amount of data being available.

1.2 A Brief History of Federated Learning

1.2.1 Seminal Work

Federated Learning as a concept was introduced in [77] which proposed an alternative to training on a single data center. It leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally-computed updates in a decentralized manner. Their method is based on iterative model averaging and conducts an extensive empirical evaluation to demonstrate that the approach is robust to the unbalanced and non-IID data distributions.

It was also used in the same year in [78] to train large recurrent language models with guarantees of user-level differential privacy. This paper also described how user privacy would be preserved in the 'large step' update of the federated averaging algorithm [77] and serves as another seminal paper in the field of Federated Learning.

1.2.2 Optimization Algorithms

Federated Learning algorithms have to account for the fact that communication with edge devices takes place over unreliable networks with very limited upload speeds. Since communication in federated learning is much more expensive than computation, it is crucial to minimize communication. It is also a kind of distributed learning [89], with the differences being that the assumptions of data being spread evenly across clients, being independent and identically distributed (i.i.d) samples from the overall distribution, and about the number of clients being much smaller than the average number of locally available training examples per client cannot be made.

That said, optimizations that apply to distributed learning are pertinent here as well. The first relevant optimization in this regard comes before [77] and is outlined in [62] where the focus lay in communication efficiency. The algorithm proposed here showed encouraging results and paved the path for future work on this topic. This was taken forward by [64] in which they proposed two ways of reducing up-link

communication costs, namely structured updates which directly learns an update from a restricted space parametrized using a smaller number of variables, e.g. either low-rank or a random mask; and sketched updates, where a full model update is learned and then compressed using a combination of quantization, random rotations, and sub-sampling before sending it to the server. Another recent work [75] reduces communication overload by identifying irrelevant updates and precludes them from being uploaded to the global model which according to their research reduces the communication footprint by 13.97 times.

In [96], the authors suggest that Multi-Task Learning is naturally geared towards handling the statistical challenges of a federated learning setting, which are to collect data in a non-IID manner across the network with the data on every node being generated by a distinct distribution. They propose a new system-aware optimization method called MOCHA, which achieves significant speed-ups compared to its alternatives as demonstrated on real world federated datasets. Another paper that got published in December of the same year, [41] focuses primarily on differential attacks on data privacy that the optimization algorithms are vulnerable to, which could originate from any party contributing during federated optimization. In such an attack, a client's contribution during training and information about their data set is revealed through analyzing the distributed model. They tackle this problem and propose an algorithm for client sided differential privacy preserving federated optimization. The aim is to hide clients' contributions during training, balancing the trade-off between privacy loss and model performance. This is yet another addition to the advances made in preserving privacy outlined in [78] and the paper on Secure Aggregation [14].

A scalable production system was recently provided by [13] in the domain of mobile devices which was based on Tensorflow. A fairly comprehensive analysis on three Federated Learning algorithms; Federated Averaging (FedAvg), Federated

Stochastic Variance Reduced Gradient, and CO-OP can be found in [85] and it shows that Federated Averaging performs the best regardless of the partitioning of the data.

However, Federated Averaging is not without its flaws. [93] casts aspersions on our understanding of Federated Averaging particularly when considering data heterogeneity across devices in terms of sample sizes and underlying data distributions. It therefore suggests an improved version of it, FedProx, that can be understood and analyzed with more veracity. Another paper [70] also states that Federated Averaging lacks theoretical guarantees and goes on to conduct experiments that indicate that the heterogeneity of data slows down convergence.

Federated Learning has been combined with Reinforcement Learning to make Federated Reinforcement Learning in the past by [112] and [73]. Recent efforts to this end were also seen at the latest IEEE INFOCOM.

1.2.3 Adversarial Work

Quite a bit of work has been done in the field of machine learning in the past on data poisoning [12], [92], [80], [103], [53], [61], [18]. Adversaries posing as honest clients can send erroneous updates that maliciously influence the properties of the trained model. This is a process known as model poisoning. In Federated Learning, [5] presents a new model poisoning strategy based on model replacement where the attacker can cause the global model to incorrectly reach a 100% accuracy.

Another recent work is by [11] which explores the threat of model poisoning attacks instead of data poisoning attacks on federated learning initiated by a single, non-colluding malicious agent. Number of strategies to carry out this attack are explored, starting with simple boosting of the malicious agent's update to overcome the effects of other agents' updates. To increase attack stealth, they have an alternating minimization strategy, which also optimizes for the training loss and the adversarial objective. Results indicate that even a highly constrained adversary can

carry out model poisoning attacks while simultaneously maintaining stealth, thus highlighting the vulnerability of the federated learning setting and the need to develop effective defense strategies.

In the presence of sybils [27], where an attacker assumes a large number of fake identities to derail the algorithm, the attack is even worse. A strategy has been suggested in [37] called FoolsGold, in which a defense that is especially effective against sybils identifies the intruders based on the diversity of client updates in the distributed learning process.

1.2.4 State-of-the-art

NeurIPS 2019 saw some interesting papers that were presented in this topic. [88] tries to augment the process of collaborative model-building. Each participating party has an independent set of labeled training examples that they wish to keep private that is drawn from a party-specific domain distribution. These users collaborate to build a general model for the task but maintain private, domain-adapted expert models. The final predictor is a weighted average of the outputs from the general and private models which they claim improves model accuracy for all users. [71] claims to improve Federated Learning by adding the functionality of learning local representations on each client device and makes a case for why this makes for a better Federated Averaging algorithm, which they call LG-FedAvg.

[105] states that the current way of performing Federated Averaging in multiple steps results in gradient biases and proposes an unbiased gradient aggregation algorithm. [101] aims to speed up the process of Decentralized Stochastic Gradient Descent with MATCHA (Matching Decomposition Sampling), which is a decentralized SGD. This is yet another effort at reducing communication delay. Another research work that's focused on improving the performance of SGD is [59] by providing values of the optimal step size and the optimal number of local iterations.

Another interesting paper that NeurIPS 2019 saw, [42] uses active learning to perform a selection on clients with a probability conditioned on the current model and data to maximize the efficiency instead of selecting clients uniformly at random.

Meanwhile, [16] provides a framework to ensure that advancements in Federated Learning adhere to certain realistic benchmarks with the help of a means of evaluation and a set of reference implementations.

1.3 Federated Learning as a Process

In this section we give a brief overview of how Federated Learning works in practice. As in supervised learning, we have

$$\text{Inputs: } x_1, x_2, x_3, x_4 ..... x_n$$

$$\text{Outputs: } y_1, y_2, y_3, y_4 .... y_n$$

Data is typically sampled from some underlying distribution that is not known to us. The objective of supervised learning algorithms is to find a function that maps from the given inputs to outputs sampled from this distribution. To do this, we decide on the form of the function and then optimize the variable parts of it. This could mean fixing the function to be linear and then finding the best possible coefficients. Because only some example data is available, the goal is to find a function that maps well from the example inputs to the outputs under the constraint that it also needs to work as well for new data sampled from the same distribution.

For another paradigm of Machine Learning, Unsupervised Learning, the outputs might be missing and the performance of the algorithm is measured differently [7]. Parts of the function which are variable and can be optimized during the learning process are called parameters or weights. Values that need to be set before the training begins are called hyperparameters.

In Federated Learning, the explicit requirement is that users do not have to share their data. The data points (n) are partitioned across the computers of k

users. Users can have varying numbers of data points and n_i denotes how many examples the i-th client has access to. Most machine learning algorithms work in an iterative process where they repeatedly look at example data. They start off with an initial solution and then continually try to improve it. In each iteration, the model is evaluated using the training data and then updated. Each update is meant to improve the model's performance on the training data a little bit.

Federated Learning functions by training on local devices which is the only instance when the data is directly being interacted with. The updates from this training are sent to the server. In some use cases of dealing with large amounts of data, it might also be cheaper to send parametric updates rather than the actual data so this is a benefit for privacy as well as communication cost.

While the edge devices usually have lower computing power than servers in a data center, there is also less data on them. By having a lot of users, the computations that need to be performed are vastly distributed. There is not much work to do on the end device of each individual. Conventional machine learning can be seen as a centralized system where all the work is performed on one server. In the process described so far, responsibilities are moved from the server to the clients. This is not a fully decentralized system because the server still runs a part of the algorithm. Instead, a federation of clients takes over a significant amount of work but there is still one central entity, a server, coordinating everything. Hence the name, *federated learning*.

Before the server starts the learning process, it needs to initialize the model. Theoretically, this can be done randomly but it makes more sense to initialize them with sensible values. If some data is already available on the server, it can be used to pre-train the model. In other cases, there might be a known configuration of model parameters that already leads to acceptable results. Having a good first model gives the training process a head-start and can reduce the time it takes until convergence.

The steps are as follows:

- At the beginning of an iteration, a subset of K clients are randomly selected by the server. They receive a copy of the current model parameters θ and use their locally available training data to compute an update. The update of the $i-th$ client is denoted by H_i
- The updates are then sent back to the server.
- The server waits until it has received all updates and then combines them into one final update.
- A new iteration begins after every model update
- In each iteration only K users are queried for updates. While requesting updates from all users would lead to more stable model improvements, it would also be extremely expensive to do because there can be millions of users. Only querying a subset of them makes it more feasible to efficiently run many iterations.
- Repeat training process until convergence

1.3.1 Comparisons to Distributed Learning

This process of Federated Learning that was outlined in the previous section can be likened to a distributed learning process in a data center. Federated Learning is more similar to a MapReduce Process [24] where computation is done in a separate and parallel manner and then combined.

There are key differences among Federated Learning and the other techniques, though, which can be outlined as follows:

- **Volume**: Typically, millions of users would be performing model updates as compared to the thousands of nodes in data centers

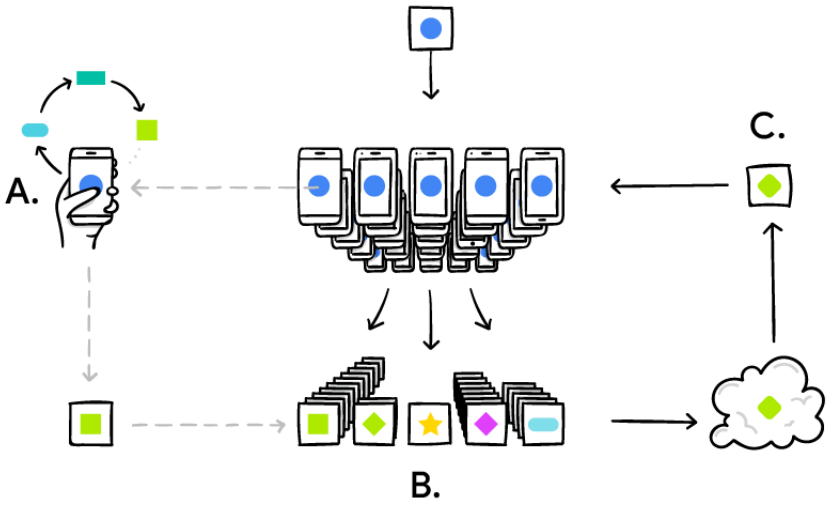

Figure 1.1 Training process in federated learning (courtesy: Google AI Blog)

- **Unbalanced Data**: Every user might have different number of parameters
- **Unpredictable and unknown data distributions**: Assumptions about underlying data distributions cannot be made about the users data.
- **Communication Issues**: Updates are sent irregularly and the scale of the communication grows proportionally with the size of the model. Compression techniques need to be employed which does not adversely affect model performance.

1.4 Use Case of Interest in Federated Learning

When the data is too private to be transferred to a server, Federated Learning is a good choice. This is often the case in situations where users implicitly generate a lot of data, just by interacting with their device. In the best case, they also label the data in this process.

An example for such an application is recommending suggestions for the next word users might want to type on a mobile phone, a functionality offered by most

virtual keyboards. Recurrent Neural Networks [21] are the go-to choice for an application like this, typically. They try to predict the next word that is going to be typed by analyzing the previously typed words. Such a model could be trained on any language corpus, for example on text from Wikipedia. However, the language used on Wikipedia differs from the one used by people in daily life. For this reason, directly training on the text typed by users would lead to better results.

Conventional methods cannot be used to do this since the data is extremely private. It should not be sent to a server and should not even be stored unnecessarily for a longer time. Federated Learning can, however, be used to train on this data because it does not compromise the data. People generate data points by typing on their devices and label these themselves as soon as they type the next word. The model can improve itself on the fly. While a user is typing, the model tries to predict the next word. As soon as the user finished typing the word, the correct label is available and the model can use this information to compute an update to improve itself. The data used to generate the update is directly discarded afterwards. This way, a high-quality model can still be trained, without making any sacrifice on privacy. Naturally, recommender system shines as a great use-case for federated learning and that is what we will be focusing on in the later chapters.

1.5 Pertinent Related Work

The work in this thesis lies in the confluence of three major fields - federated learning, recommender systems and cryptography, specifically homomorphic encryption. In this section, we take a look at the related works that also combine elements of this research.

1.5.1 Federated Learning and Recommender Systems

[17] incorporates meta learning, a machine learning paradigm where a top-level (or meta-level) artificial intelligence optimizes a bottom-level artificial intelligence, into a federated learning framework. The model is distributed and trained on the clients and the server maintains a shared model by aggregated updated models from the clients. They use this framework for making recommendations for edge device users. They build upon [33] and propose *FedMeta*. One of their experiments is on an industrial recommendation task and they outperform stand-alone models as well as models trained by a typical federated learning approach.

[1] was the first paper in the space of federated collaborative filtering, which our this work is inspired by. They utilize deep learning models to train a collaborative filtering recommendation system that models the interactions between a user and a set of items. The item factor vectors are updated on the server and distributed to each client and local user updates take place on client devices. They use an alternating least squares as well as gradient descent to perform model updates along with Adaptive Moment Estimation (Adam) for their Federated Collaborative Filter. The test is conducted on MovieLens' 25 million movie dataset.

[54] is a federated collaborative filtering system that cites [33], [84] and [1] as influences and proposes a system that is related to [84], but unlike the Reptile meta-learning algorithm, their approach does not require the edge devices to run sequentially and update the central parameters in turns, or communicate with each other. Instead each device communicates with the parameter server independent of each other the training takes place in parallel for every device. In [34] presents a matrix factorization technique for taking into consideration multi-view structure of the data.

1.5.2 Federated Learning and Homomorphic Encryption

[49] introduces the concept of vertical federated learning, where data is split by features and only one data provider knows the target variable. They utilize concepts of additively homomorphic encryption [81] for security from an honest-but-curious adversary and perform entity resolution. [20] provides a tree boosting system in the context of federated learning that is lossless and privacy preserving. They use homomorphic encryption extensively and provide a mathematical proof of security. [99] tries to introduce a method that combines secure multiparty computation and differential privacy in a bid to create a hybrid approach to privacy-preserving federated learning. This aims at reducing the shortcomings of both the approaches, achieves a high level of accuracy and is a scalable approach to combat inference threats. They claim that the use of homomorphic encryption in their scheme greatly increases the accuracy. [38] also utilizes homomorphic encryption in their system that introduces a secure federated transfer learning. In addition, [102] also discusses protecting against user level privacy leakage using homomorphic encryption techniques.

1.5.3 Homomorphic Encryption and Recommender Systems

[30] and [107] were among the first to introduce the frameworks to combine additively homomorphic encryption [81] and recommender systems to create a privacy enhanced recommender systems and proposed systems in which service providers learns no information on any user's preferences. This was suggested for a centralized system and the idea was to generate recommendations by processing them under encryption. A more efficient way of doing this was introduced in [32] by the same authors two years later. The same year they also released a paper that performed content-based filtering in a secure manner using homomorphic encryption [31]. More recently [58] used homomoprhic encryption for a privacy preserving healthcare recommender system

that uses collaborative filtering and [4] also proposes a privacy preserving user-based recommender system

CHAPTER 2

BACKGROUND

2.1 Estimates, Bias and Variance

It is often better to compute estimates rather than exact values. Estimation Theory [26] provides adequate formalism for the same. An *estimator* is any decision rule or a function that estimates a value based on certain observations, random or otherwise. There are several ways to measure the quality of an estimator:

$$bias[\tilde{X}] = E[\tilde{X}] - X \tag{2.1}$$

If the bias of an estimator is 0, it is called an unbiased estimator. This typically means that the estimator is generally correct on average. If one samples for long enough from the estimator, the average converges to the true value X. This is due to the law of large numbers.

If an estimator is unbiased, its individual estimates can still be far off from the true value. While the mean of many sampled estimates eventually converges to the true expected value, this can take a long time, meaning the estimator is inefficient. To quantify how consistently an estimator is close to the true value, another statistic is required. Commonly, the variance of the estimator is considered here. It is defined as the mean squared distance between the estimate and the value to be estimated.

$$Var[\tilde{X}] = E[\tilde{X} - X]^2 \tag{2.2}$$

A lot of different things can be analyzed using estimators. For example, statistical models can be seen as estimators. They use observations, or data, to make predictions. These predictions are generally not perfect because randomness is

involved and only a limited amount of information is available. Thus, it makes sense to analyze statistical models in terms of bias and variance. A very usual problem when building models is balancing underfitting and overfitting. If the training data is just memorized, the model does not generalize well to new data, which is called overfitting. The flip side of the issue is when it the estimated curve barely matches the pattern in the training data, which is referred to as underfitting.

This family of problems is known as bias-variance tradeoff. If the model has a high bias, its predictions are off, which corresponds to underfitting. If overfitting occurred, that is, the data is matched too well, the estimates have a high variance. By resampling the data that the model was built on, different estimates are generated because the model is now based on different random noise. Optimizing both bias and variance is usually unrealistic for statistical models, so a balance (or a trade-off) needs to be reached. Depending on how the estimators are used, different qualities are given more importance over others.

2.2 Gradient Descent

Machine learning can be considered an optimization problem. In supervised learning, a loss function quantifies how close a prediction $f(x_i)$ of a model is to the correct answer y_i. The parameters of the model should be chosen to minimize the loss. Generally this is done by optimizing them for the training data while also validating the model on otherwise unused data. The parameters with the best performance on the validation data are selected in the end. The loss for a prediction function f is given by:

$$L = \frac{1}{n} \sum_{1}^{n} loss(f(x_i), y_i) \tag{2.3}$$

The Squared Error function serves as a popular choice for the loss function:

$$loss(a, y) = (a - y)^2 \tag{2.4}$$

There are many algorithms centered around minimizing loss. A huge part of machine learning is based on a conceptually simple method called gradient descent. It is an iterative algorithm where the model parameters are repeatedly moved into a direction where they work a little bit better.

The model is initialized with an arbitrary set of parameters and then the iterative optimization process begins. In each iteration, the partial derivatives of the loss with respect to the model parameters are computed. To be able to do this, gradient descent requires the prediction function f and the loss function to be differentiable. An important property of the vector of partial derivatives is that it points into the direction of steepest ascent. Because the loss should be minimized, the parameters are updated into the opposite direction. Since the vector of partial derivatives only points to the next optimum but does tell us how far to go, a scaling factor η (eta), called the learning rate, is also applied.

Each parameter θ_i is then repeatedly updated using this idea:

$$\theta_i \longleftarrow \theta_i - \eta * \frac{\partial L}{\partial \theta_i} \tag{2.5}$$

In each iteration, all parameters are updated at the same time using this formula. The parameters are repeatedly improved, getting closer a local optimum. Once a local optimum is reached, the gradient becomes 0 and no more updates are performed. In practice, the learning rate needs to be tuned well to make sure that the updates do not jump over an optimum.

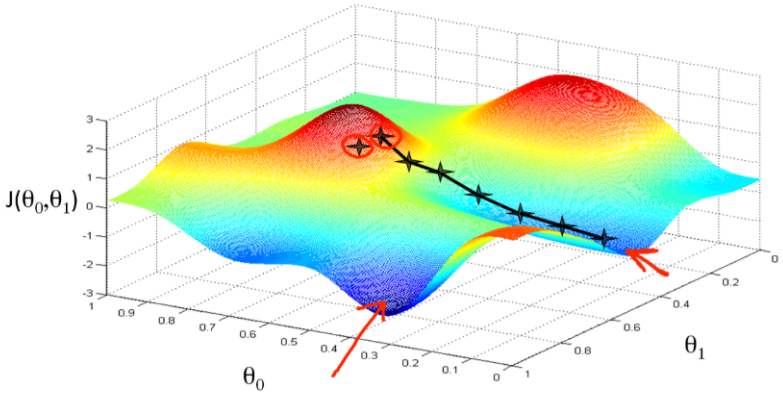

Figure 2.1 A 3D rendition of gradient descent (courtesy: HackerNoon)

2.2.1 Stochastic Gradient Descent

A simplification of the partial derivatives notation can be done by using the gradient operator ∇. This can be used by assigning the operator ∇ as the vector of all partial derivatives as follows:

$$\nabla_i L = \frac{\partial L}{\partial \theta_i} \tag{2.6}$$

To actually calculate the gradient ∇ of the loss, one can make use of the linearity of the gradient operator:

$$\nabla L = \nabla \frac{1}{n} \sum_{i}^{n} loss(f(x_i), y_i) \tag{2.7}$$

$$= \frac{1}{n} \sum_{i}^{n} \nabla loss(f(x_i), y_i) \tag{2.8}$$

n is typically a very large value which makes for some unnecessarily laborious and resource-intensive calculations. For the purposes of saving time, a part of the data can be used to estimate the gradient. In stochastic gradient descent (SGD), a

single data point x and label y are sampled uniformly from the training set. The true gradient ∇L is estimated using:

$$\nabla \tilde{L} = \nabla loss(f(x), y) \tag{2.9}$$

It can be proven that the SGD estimate $\nabla \tilde{L}$ is an unbiased estimator of ∇L [98]

The computations for Stochastic Gradient Descent can be performed very quickly but still give us an unbiased estimate of the true gradient. This property is the reason why optima can be found using this algorithm. While individual estimates are off, the randomness averages out over iterations and the parameters still move into a sensible direction overall. Since iterations are much cheaper, more of them can be performed. The framework utlized in this thesis uses a lot of Stochastic Gradient descent for this purpose.

2.3 Mini-Batch Gradient Descent

Individual estimates can have a large variance, leading to noisy and jumpy updates. An improvement over the Stochastic Gradient Descent is the mini-batch gradient descent. Instead of just sampling one data point, small samples of k data points are sampled. The estimated gradient is an average of all k single estimates. Each of these individual estimators is unbiased. Thus, their average also has to be an unbiased estimator. In contrast to SGD however, there is much less variance, because more data is used to compute the estimate. Most gradient computations can be formulated using linear algebra operations which are highly parallelizable using GPUs. With appropriate hardware there is no significant performance penalty for using 1 ¡ k ¡¡ n data points to compute the estimate. Because the variance is much smaller compared to SGD, mini-batch gradient descent typically has the best convergence rate

in practice. Gradient descent can only guarantee the convergence to local optima. Gradient descent can also get stuck in saddle points and plateaus.

Plateaus commonly occur when the gradient is dominated by one dimension, in which case updates mostly slide along this one dimension. If there is little improvement of the loss in this dimension, then the algorithm gets stuck on a plateau. A saddle point has less requirements than an optimum. While the gradient is 0 in all directions, it does not mean that there is no better point in the close neighborhood.

2.4 Big Data Challenges

Rich sources of data are found in the mobile devices we use everyday, be it phones, tablets, or automotive vehicles. These devices are equipped with a plethora of different sensors capable of producing vast amounts of data each day. For example, a moderately sized fleet of 1000 test vehicles is estimated to be able to produce data in the order of terabytes each day [55] and Hitachi claims that a modern internet connected vehicle produces data in excess of 25 GB/hour. This data abundant setting is often referred to as big data.

Big data analytics holds promise for more accurate validation and testing models, as well as improving consumer experience. The SGD algorithm requires that all training examples are available on the machine that trains the ANN. This centralised approach has limitations in a big data setting, where data is generated in huge volume and with great velocity. Notably, if data is generated on edge devices, then wireless data transfer over cellular networks becomes the principal bottleneck, which suggests that transferring all generated data is infeasible in practice. In addition, if attempted we would quickly exhaust the consumers' data plans in the case of phones. Besides the challenge of data transfer, simply storing such vast amounts of data centrally will quickly become a problem in itself. Besides the technical

challenges posed by big data, collecting and storing large amounts of data centrally is problematic from a privacy perspective.

The General Data Protection Regulation (GDPR) recently came into effect in the EU, which has several implications for how personal data can be collected and processed. Data regarding, for instance, driving patterns is indeed personal if it can be connected to an individual in any way. GDPR has principles for purpose and storage limitation as well as data minimization. Personal data can only be processed with explicit consent for specific, known purposes (purpose limitation), and only the minimal amount of data required to fulfil those purposes may be collected (data minimisation). Moreover, data should be removed once its purpose has been fulfilled (storage limitation). Performing big data analytics on consumer data while respecting data privacy laws is therefore not a straightforward process.

2.5 Optimization Problem Formulation

The optimization community has seen an explosion of interest in solving problems with finitesum structure in recent years. In general, the objective is formulated as

$$\min_{w \in R^d} P(w) \tag{2.10}$$

where

$$P(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{2.11}$$

The main source of motivation are problems arising in machine learning, typically in linear or logistic regressions and support vector machines.

More complicated non-convex problems arise in the context of neural networks, where rather than via the linear-in-the-features mapping, the network makes predictions through a non-convex function of the feature vector x_i . However, the resulting loss can still be written as $f_i(w)$, and gradients can be computed efficiently

using backpropagation. The amount of data that businesses, governments and academic projects collect is rapidly increasing. Solving the optimization problem in practice is often impossible on a single node, as merely storing the whole dataset on a single node becomes infeasible. This necessitates the use of a distributed computational framework, in which the training data describing the problem is stored in a distributed fashion across a number of interconnected nodes and the optimization problem is solved collectively by the cluster of nodes. One can use any network of nodes to simulate a single powerful node, on which one can run any algorithm. The practical issue is that the time it takes to communicate between a processor and memory on the same node is normally many orders of magnitude smaller than the time needed for two nodes to communicate; similar conclusions hold for the energy required [94].

Further, in order to take advantage of parallel computing power on each node, it is necessary to subdivide the problem into sub-problems suitable for independent/parallel computation. State-of-the-art optimization algorithms are typically inherently sequential. Moreover, they usually rely on performing a large number of very fast iterations. The problem stems from the fact that if one needs to perform a round of communication after each iteration, practical performance drops down dramatically, as the round of communication is much more time-consuming than a single iteration of the algorithm. These considerations have lead to the development of novel algorithms specialized for distributed optimization.

2.6 Distributed Optimization

When the number of training examples becomes too large to store on one computer, which is the case in a big data context, we have to distribute the computation to multiple computers. Distributing the data and computational burden to multiple computers leads us to reformulate the loss function. Assume there are K clients to

which data and computation are distributed. Each client then holds a part P_k of all training examples, and computes $F_k(w)$, which is the average loss on client k. If the number of training examples held by client k is denoted by $n_k = |P_k|$, then we can rewrite the objective function as a weighted sum over all $F_k(w)$:

$$f(w) = \sum_{i}^{n} \frac{n_k}{n} F_k(w) \tag{2.12}$$

However, existing approaches focus mainly on the case of data centre optimisation where computation rather than communication is the primary bottleneck. Distributed data centre optimisation typically requires control over the data distribution since these approaches rely on balanced, i.e. equal n_k for all k, and IID data assumptions. These assumptions are too strong when we want to perform learning tasks on a heterogeneous ecosystem of edge devices.

When we speak about solving the optimization problem in a distributed setting, we refer to the case when the data describing the functions f_i are not stored on any single storage device. This can include setting where one's data just don't fit into a single RAM/computer/node, but two is enough. This also covers the case where data are distributed across several data centers around the world, and across many nodes in those data centers. The point is that in the system, there is no single processing unit that would have direct access to all the data. Thus, the distributed setting does not include single processor parallelism. Compared with local computation on any single node, the cost of communication between nodes is much higher in terms of both speed and energy consumption, introducing new computational challenges, not only for optimization procedures. The communication efficient algorithms provide us with much more flexible tools for designing overall optimization procedure, which can make the algorithms inherently adaptive to differences in computing resources and architectures.

This setting creates unique challenges. Distributed optimization algorithms typically require a small number (1–4) of communication rounds per iteration. By a communication round we typically understand a single MapReduce operation [24], implemented efficiently for iterative procedures, such as optimization algorithms. Spark [106] has been established as a popular open source framework for implementing distributed iterative algorithms, and includes several of the algorithms mentioned in this section. Recent decade has seen an explosion of interest in distributed optimization, greatly motivated by rapid increase of data availability in machine learning applications. Much of the recent effort was focused on creating new optimization algorithms, by building variants of popular algorithms suitable for running on a single processor. A relatively common feature of many of these efforts is that the computation overhead in the case of synchronous algorithms, and the difficulty of analysing asynchronous algorithms without restrictive assumptions. By computation overhead we mean that if optimization program runs in a compute-communicate-update cycle, the update part cannot start until all nodes finish their computation. This causes some of the nodes to be idle, while remaining nodes finish their part of computation which is clearly an inefficient use of computational resources. This pattern often diminishes or completely reverts potential speed-ups from distributed computation. In the asynchronous setting in general, an update can be applied to a parameter vector, followed by computation done based on a now-outdated version of that parameter vector. Formally grasping this pattern, while keeping the setting realistic is often quite challenging. As a result, this is an open area of research, and optimal choice of algorithm in any particular case is often heavily dependent on the problem size, details in its structure, computing architecture available, and above all, expertise of the practitioner. This general issue is best exhibited with numerous attempts at parallelizing the Stochastic Gradient Descent and its variants.

CHAPTER 3

BUILDING BLOCKS OF FEDERATED LEARNING

3.1 Federated Optimization

Federated Learning can be optimized in many ways, for instance, a mechanism for dealing with when the servers are idle. This is a case in distributed systems that is unique to federated learning. Communication efficiency and reducing the number of communication rounds is also what Federated Optimization deals with and can be viewed as an extension of Distributed Optimization. One of the most important works that has been done in this field is [62].

3.2 Distributed Training Data

The Federated Learning protocol described earlier implements an adapted form of mini-batch gradient descent. To estimate the full update more efficiently, only some data points are used in each iteration. But since the data points are partitioned across users, they are not sampled completely independently of each other anymore.

Consider a simplified protocol where in each iteration only a single user is sampled. The i^{th} user is selected with a probability of $\frac{n_i}{n}$. Their j^{th} data point and label are denoted by x_{ij} and y_{ij} respectively. The full update proposed by the i^{th} user is then given by

$$H_i = \sum_{j=1}^{n} \frac{1}{n_i} \nabla loss(f(x_{ij}), y_{ij}) \tag{3.1}$$

Even though the data is arbitrarily distributed among users, an unbiased estimate can still be computed. The final estimate in that case is an average of the individual estimates, weighted by the number of data points that the clients used. While the distribution of the training data has no influence on the bias of the

estimator, it can have a huge effect on the variance. If it can be controlled where the data is located, it is easy to ensure that all users produce similar estimates. In that case, the variance of the estimator is low and convergence can be reached quickly.

In Federated Learning, however, such control over the data is not possible and the variance can be large. As an extreme example, consider a situation where users only have two kinds of data points, which are extremely different from each other. If the location of training data can be changed, like in a data center, each computing node can have a similar number of both data points. In Federated Learning, it is possible that users either only have the first kind of data points or only the second. In that case, the two kinds of users will produce very different gradient estimates, leading to a larger variance.

3.3 Algorithms for Federated Optimization

This section discusses algorithms made keeping distributed optimization in mind and finally discussed a technique that combined them that is used for Federated Optimization. There are two seemingly unrelated algorithms, one for distributed optimization and another for communication efficiency that are discussed, and their connection forms the bulk of [63]'s research. The algorithms are the Stochastic Variance Reduced Gradient (SVRG) [56] a stochastic method with explicit variance reduction, and the Distributed Approximate Newton (DANE) [95] for distributed optimization. The descriptions are followed by their connection which gives rise to Federated SVRG (FSVRG).

Certain properties one would hope to find in an algorithm for the non-IID, unbalanced, and massively-distributed setting are worth considering here. These are

- If the algorithm is initialized to the optimal solution, it stays there.
- If all the data is on a single node, the algorithm should converge in O(1) rounds of communication.

- If each feature occurs on a single node, so the problems are fully decomposable (each machine is essentially learning a disjoint block of parameters), then the algorithm should converge in O(1) rounds of communication)

- If each node contains an identical dataset, then the algorithm should converge in O(1) rounds of communication.

For convex problems, convergence has the usual technical meaning of finding a solution sufficiently close to the global minimum, but these properties also make sense for non-convex problems where convergence can be read as "finds a solution of sufficient quality".

In these statements, $\mathcal{O}(1)$ round is ideally exactly one round of communication. The first property is valuable in any optimization setting. The second and third properties are extreme cases of a federated optimization setting (non-IID, unbalanced, and sparse), whereas the fourth property is an extreme case of a classic distributed optimization setting (large amounts of IID data per machine), and therefore is not relevant for consideration in a federated optimization setting.

Here is what SVRG (Stochastic Variance Reduce Gradient) looks like algorithmically:

Algorithm 1 Stochastic Variance Reduced Gradient

parameters: m = #stochastic steps per epoch, η = step length

for $s = 0, 1, 2...$ **do**

  Compute and store $\nabla P(w^t) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w^t)$ ▷ A full pass through data

  $w = w^t$

  for $t = 1$ to m **do**

    Pick i ∈ 1, 2, . . . , n, uniformly at random

    $w = w - \eta(\nabla f_i(w) - \nabla f_i(w^t) + \nabla P(w^t))$ ▷ Stochastic Update

  $w^{t+1} = w$

The update is done according to the rule:

$$w^t = w^{t-1} - \eta(\nabla f_i(w) - \nabla f_i(w^t) + \nabla P(w^t)) \tag{3.2}$$

The algorithm runs in two nested loops. In the outer loop, it computes gradient of the entire function P (Line 3). This makes a full pass through data which is usually an expensive task one tries to avoid unless necessary. This is followed by an inner loop, where fast stochastic updates are performed. m is typically set to be a small multiple (1–5) of n. Although the theoretically optimal choice for m is a small multiple of a condition number, this is often of the same order as n in practice. The central idea of the algorithm is to avoid using the stochastic gradients to estimate the entire gradient $\nabla P(w)$ directly.

in the stochastic update in Line 7, the algorithm evaluates two stochastic gradients, $\nabla f_i(w)$ and $\nabla f_i(w^t)$. These gradients are used to estimate the change of the gradient of the entire function between points w t and w, namely $\nabla P(w) - \nabla P(w^t)$. Using this estimate together with $\nabla P(w^t)$ pre-computed in the outer loop, yields an unbiased estimate of $\nabla P(w)$. Apart from being an unbiased estimate, it could be intuitively clear that if w and w t are close to each other, the variance of the estimate $\nabla f_i(w) - \nabla f_i(w^t)$ should be small, resulting in estimate of $\nabla P(w)$ with small variance. As the inner iterate w goes further, variance grows, and the algorithm starts a new outer loop to compute new full gradient $\nabla P(w^{t+1})$ and reset the variance.

The DANE (Distributed Approximate Newton) algorithm [95] is a distributed Newton-type method aimed improving communication efficiency where in each iteration there are two rounds of communication. It was proposed to exactly solve a general subproblem available locally, before averaging their solutions. The method relies on similarity of Hessians of local objectives, representing their iterations as an average of inexact Newton steps.

These algorithms led [63] to write Federated SVRG, the algorithm for which is as follows:

Algorithm 2 Federated SVRG

parameters: h = stepsize, $\mathcal{P}_{k=1}^{K}$, diagonal matrices $A, S_k \in \mathcal{R}^{d \text{ x } d}$

for $s = 0, 1, 2...$ **do** ▷ Iterations

  Compute $\nabla f(w^t) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w^t)$ ▷ Distributed Loop

  for $k = 1$ to K **do**

    Initialize" $w_k = w^t$ and $h_k = h/n_k$

    Let $i_t^{n_k}$be random permuation of $\mathcal{P}_k$

    for $t = 1$ to n_k **do**

      $w_k = w_k - h_k(S_k[\nabla f_{i_t} w_k - \nabla f_{i_t} w^t] + \nabla f(w^t))$

  $w^t = w^t + A \sum_{k=1}^{K} \frac{n_k}{n} (w_k - w^t$

This was the first implementation of an algorithm keeping Federated Optimization in mind and was further expanded upon by [69].

3.4 Compression

3.4.1 Communication Cost

A naive implementation of the previously introduced Federated Learning protocol scales badly when used with more complex models. This is because the protocol requires a lot of communication between clients and the server. In each iteration, sampled clients download a copy of the current model and upload updates for all its parameters. This has the effect that the amount of required communication grows proportionally with the size of the model. In the past few years, models have become much larger, with neural networks consisting of more and more layers. It is not uncommon anymore to use neural networks with millions of parameters. Overviews of models have shown that the number of parameters, and with them the total computational cost, of the largest models has doubled every three to four months.

This differentiates the techniques in this section from standard compression techniques. Compressing the entire model for downloads is a another very different problem, where the goal is to keep the predictions as stable as possible [46]. This can however significantly decrease the quality of the model, and might thus not always be an option. From a theoretical perspective, we would like to reduce the communication requirements for uploads by more than a constant factor. This will ensure that updates can still be transmitted efficiently , even if the bandwidth of network connections does not increase as quickly as the size of common models in the future. From a practical standpoint, this is not entirely necessary. Decreasing the size of uploads by an order of magnitude makes uploads as efficient as downloads. Since compressing the model itself for downloads might not be desirable, a constant compression factor for uploads can already be sufficient.

3.4.2 Structured and Sketched Updates

A good way to represent very large data sets is as a matrix. For instance, if there are n data points, and each data points has d attributes, then this can be thought of an nxd matrix A with n rows and d columns. While matrix approximation and decomposition has been studied in numerical linear algebra for many decades, these methods often require more space and time than is feasible for very large scale settings, and also often worry about more precision than is required. The last decade has witnessed an explosion of work in matrix sketching where a more sparse input matrix is efficiently approximated with a more compact matrix (or product of a few matrices) so that the resultant matrix preserves most of the properties of the input up to a guaranteed approximation ratio.

Individual compression can be very lossy in nature. While it is not possible to completely get rid of an error without switching to lossless compression algorithms, we still want to ensure that the compressed update leads to a good model in the

immediate future. Structured update techniques enforce a certain form on the update that allows for a more efficient representation. The fact that the update must have a structure is considered during the optimization process itself and by doing this, the optimizer can find updates that follow the enforced structure but still minimize the loss. In other words, we can find a locally optimal update that can be encoded efficiently, since it has the predefined structure. In this process, the theoretical guarantees of unbiased estimators are lost, and the methods are only motivated on a heuristic level.

A structured update method allows us to change the optimization process itself. To this end, we perform several steps of SGD based on the sparse updates. After the updates of the first step were computed, they are applied locally and new sparse updates are computed. Only the weights chosen by the mask are updated in all these steps. Another method for enforcing a structure on the updates in a way that allows for an efficient encoding is based on matrix decomposition. Instead of directly sending a matrix H, we find two matrices A and B that can be used to approximate H. Concisely, the update matrix can be factorized using A and B:

$$H = A * B \tag{3.3}$$

Sketched update methods in Federated Learning first learn the full update and then compress it. Unbiased estimators can be used to efficiently approximate some values. This concept can also be applied to the compression of updates. As mentioned previously, sketched update methods want the compressed update H to be an unbiased estimator of the true update H. This means that the compressed update is the true update on average, even though it can be encoded much more efficiently. This way we can still compute an unbiased estimate of the true gradient, just like

when only working with a subset of clients in each iteration. These methods also give us a theoretical justification of why they work.

In the sparse mask compression method for sketched updates, each client is only allowed to update a certain set of weights in every iteration. This idea can be extended into a structured update technique. If a client can only update certain weights, then it could take this information into account during the optimization process. Instead of computing a normal weight update and then making it sparse afterwards, updates should only be computed for the specific weights selected by the sparse mask. To implement this, the other weights are marked as constants in the computational graph. When computing the updates like this in one SGD step, the results are not as different compared to the ones obtained in the sketched method because in the process of computing partial derivatives, all the other weights are treated as constants.

3.5 Privacy

A primary motivator for introducing Federated Learning were privacy concerns with the way machine learning is conventionally done. Federated Learning is a major improvement in this regard since the data does not leave the users' devices anymore. However, it still remains to be discussed if it is possible to reconstruct data based on the updates that are sent to the server. Since neural networks are universal function approximators, they are able to approximate a function that acts as a look-up table to all the data [23].

Neural networks with too many neurons typically memorize parts of the training data instead of learning more general pattern [43]. We have to assume that an adversary could intercept all messages and read the model weights. This adversarial actor could analyze the weights to figure out information about individuals. Weights of neural networks have the reputation of being incredibly hard to analyze [39]. There

has been some work in this area - [36] for instance, analyzed the weights of a facial recognition model. They were able to reconstruct some of the faces that were used in the training of the model. While all of this remains incredibly difficult to do and there have been few successful such attempts, they show that attack vectors to Federated Learning do exist. More work on this was also done recently by [2], [111] and [82] who all were able to successfully infer data about clients based on model weights.

When looking at the techniques introduced so far, we are not able to quantify how difficult it really is to figure anything out about the data of individuals. Historically, there have been many cases where people tried to data make data anonymous to ensure the privacy of individuals but without formal guarantees, this anonymization often looked solid has been broken later on. One of these cases was related to the Netflix Prize [67]. Netflix published a dataset that contained information about users and which movies they liked. The dataset was meant to be used in a competition to improve the Netflix recommender system. Obvious personal identifiers, such as names and user IDs, were removed from this dataset. Many users however also published their movie reviews on IMDb. By joining the Netflix and IMDb datasets, researchers were able to deanonymize parts of the Netflix dataset [83].

A similar case occurred in the 1990s, when a government agency in Massachusetts published a dataset about hospital visits of its employees [8]. This dataset was meant for research purposes. Personal identifiers such as names and social security numbers were removed yet again, however, the dataset was still deanonymized in parts by joining it with a voter roll dataset. Datasets with information about people registered for voting can be bought legally in the US. Since both datasets shared some fields, it was possible to join them. Among those whose details were compromised was governor William Weld, whose data was possible to extract by cross-referencing gender and date of birth with the postal code of his address.

This showed that personal attributes make it surprisingly easy to identify people. In the US, 87% of people can be uniquely identified by gender, birth date and postal code alone. When the data was published, steps for anonymization were taken. However, no formal guarantees regarding privacy could be made. Formalization is desirable when it comes to privacy. While Federated Learning looks like it is much better for privacy, formal guarantees are needed which are robust enough to hold even if the attacker has unexpected access to information.

3.5.1 Differential Privacy

Most recommender systems use Differential Privacy [28] for privacy preservation. Differential Privacy is a mathematical field that uses a stochastic framework [29]. It allows us to quantify how much certain algorithms respect privacy. We do not consider algorithms to be either privacy-preserving or to be bad for privacy. Instead, the objective is to describe how difficult it is to gather information regarding the individuals concerned. For example, one might want to compute the mean value across many users without knowing the exact values of individuals. In the case of machine learning, we want to fit a model without knowing details about the data of individuals. To describe this formally, two datasets $D1$ and $D2$ are considered. These datasets are identical except for one row of data which is missing in one of them. A statistical query Q is then executed on both datasets. This query could, for example, compute the mean or fit a statistical model. It usually involves randomness. Even for queries that are not random by default, such as computing the mean, adding randomness helps to improve privacy guarantees.

A query Q is considered to be differentially private if for all adjacent datasets $D1$, $D2$ and for every possible subset R of results of the query, the following formula holds:

$$P[Q(D1) \in R] \leq e^{\epsilon} * P[Q(D2) \in R] \tag{3.4}$$

In other words, adding a new data point to a dataset must not substantially change the result of the query. The formula is expressed using probabilities to account for the randomness in Q. If we guess the results of the query to be in a set of possible values R, then adding one data point should not change the probability of being correct by more than e. If this is not the case, then adding the new data point is bad for privacy because it is noticeable whether the data point was used by the query or not. The value is called the level of Differential Privacy. The definition above captures what we intuitively understand as privacy. The point is to make it very hard to extrapolate whether an individual contributed data, and much less so what their data looks like. To make algorithms fit into the framework of Differential Privacy, randomization strategies are used where instead of having users report their true data, they only share a version with random noise was added on top. In the case of discrete data where it is hard to add a little bit of noise, users could lie with some given probability. By doing this, the entity collecting the data cannot make confident conclusions about individuals anymore. However, it is possible to estimate the overall random noise well when enough users are surveyed.

$$P[Q(D1) \in R] \leq e^{\epsilon} * P[Q(D2) \in R] + \delta \tag{3.5}$$

An additional δ is added to the previous definition, which allowed for a probability δ of directly breaking Differential Privacy. We want to keep both σ and δ small to ensure good privacy. To show that an algorithm conforms to some form of Differential Privacy, the concept of sensitivity is often used.

The sensitivity S(Q) of a query Q describes by how much the result can differ if the query is executed on two adjacent datasets:

$$S(Q) = max_{D_1,D_2}||Q(D_1) - Q(D_2)||_2 \tag{3.6}$$

where $||.||$ is the L_2 norm. Sensitivity should ideally be bounded by a constant or at the very least, be low.

We perform an analysis of differential privacy later in the thesis.

3.5.2 Encryption Techniques

No discussion about preserving privacy is complete without mentioning the work on encryption that has been carried out by computer scientists and network experts and it has been a field that has been relevant for as long as communication in the face of adversarial units has been around. Today, a lot of popular encryption techniques fall under the category of asymmetric encryption in which a public key is used to encrypt data but only someone with its specific matching private key is able to decrypt it.

We would like to introduce the encryption technique that we have used in this work, a method called homomorphic encryption [40]. Most encryption techniques being employed today are virtually impregnable, but require decryption to be used for any practical computational purposes. Homomorphic encryption is a technique that allows for computation on encryted data. The data can be accessed and processed while still in its encrypted form. A good example of performing computation on homomorphic encryption, according to its inventor, Gentry, would be that of a glove box. The analogy presented was that the operators using the glove box can operate on the materials inside the lock without ever having direct access to the raw materials inside it.

In [17], Chen discusses the computational performance when using homomorphic encryption. They find that while either fully or partially homomorphic encryption is computationally expensive, it can be implemented in a way on a large scale to not have a great deal of effect regarding performance.

A technique like this naturally lends itself as a perfect opportunity for use in machine learning and indeed Federated Learning as well [49]. Its use in this thesis will be explored in a latter chapter.

3.6 Categories of Federated Learning

[104] provides a very helpful categorization of federated learning along with a comprehensive collection of all the work that identifies as such. Here is a brief overview of the definitions.

3.6.1 Horizontal Federated Learning

This is the case for the scenarios in which datasets occupy the same feature space but have a different space for samples. In other words, when the person/organization carrying out the federated learning requires the same features from different groups of users. For instance in mobile keyboard prediction where the user base is different depending on regions but the features needed are similar.

3.6.2 Vertical Federated Learning

The scenario for datasets occupying the same sample space but having a different feature space. To put it in simpler terms, when an organization requires different features from the same group of users. For example, two separate companies operating in the same geographical region.

3.6.3 Federated Transfer Learning

Federated Transfer Learning is the case in which the datasets occupy different feature space as well as different sample spaces. An example provided by [104] for this is that suppose there are two banks operating in two different countries that speak different languages. They are likely to share only a very limited set of features and have vastly different user bases. In this case, transfer learning techniques prove conducive to using Federated Learning.

CHAPTER 4

RECOMMENDER SYSTEMS OVERVIEW

Recommender Systems, as defined by [91] are software tools and techniques providing suggestions for items to be of use to a consumer. These suggestions are a convenience tool that may be used by the consumer for making decisions. A recommender system normally focuses on a specific type of item (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations are all customized to provide useful and effective suggestions for that specific type of item. Most popular and important use cases include but are not limited to - product recommendations, movie or TV show recommendations, article recommendations and the list goes on. Facebook [57] uses a rotational hybrid approach to recommender systems which yields a very accurate result, some of which feels spooky to a lot of users. We take a look at what the traditional approaches are in this section and expound on those approaches to see how they can be utilized in a federated context in the next section.

4.1 Collaborative Filtering

Tapestry [90] was instance recommender systems being used in practice for recommending documents from newsgroups. Its authors also introduced the term collaborative filtering to help users with the large volume of documents. Collaborative filtering systems make recommendations based on historic users' preference for items. The preference can be presented as a user-item matrix. Here is an example of a matrix describing the preference of 4 users on 5 items, where a_{12} is the user 1's preference on item 2.

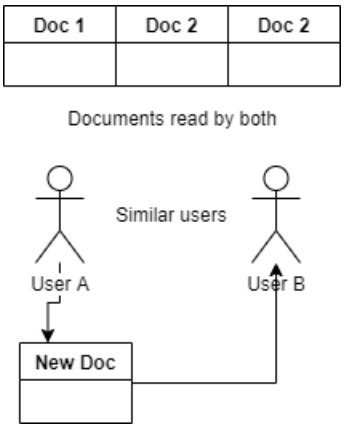

Figure 4.1 Collaborative filtering,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

For practical settings, this user-item matrix is very large and has missing preference values for users, and the goal of collaborative filtering and by extension, recommender systems is to fill these missing values in accurately.

Nearest neighbour [9] algorithm is the first approach to collaborative filtering and it is based on the similarity between pairs of items or users. For this purpose, cosine similarity is used:

$$similarity(a, b) = cos(a, b) = \frac{a \cdot b}{||a|| * ||b||} \tag{4.1}$$

There can be user based as well as item based collaborative filtering. If the preference matrix is represented by item vectors and the user matrix by user vectors, they would be

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_n \end{bmatrix}$$

$$\begin{bmatrix} I_1 & I_2 & I_3 & \dots & I_n \end{bmatrix}$$

The similarity between U1 and U2 is calculated as cos (U1,U2). The missing values in the preference matrix are commonly filled with zeros. For U_i, we can recommend the items liked by U_i's most similar users (user-to-user) or the most similar items of U_i's liked items (item-to-item).

For user-based collaborative filtering, one can have an n x m matrix of ratings, with user U_i, where $i = (1, ...n)$ and item I_j, where $j = (1, ...m)$. Now we want to predict the rating r_{ij} for the case in which the target user i did not watch or rate an item j. The process is to calculate the similarities between target user i and all other users. We do this by selecting the top x similar users, and take the weighted average of ratings from these x users with similarities as weights.

$$r_{ij} = \frac{\sum_k \text{Similarities}(U_i, U_k) r_{kj}}{\text{total ratings}} \tag{4.2}$$

While different people may have different baselines when giving ratings, some people tend to give high scores generally, some are pretty strict even though they are satisfied with items. To avoid this bias, we can subtract each user's average rating of all items when computing weighted average, and add it back for target user, shown as below

$$r_{ij} = \bar{r}_i + \frac{\sum_k \text{Similarities}(U_i, U_k)(r_{kj} - \bar{r}_k)}{\text{total ratings}} \tag{4.3}$$

Similarly, for Item-based Collaborative Filtering, we say two items are similar when they received similar ratings from a same user. We then make predictions for a target user on an item by calculating weighted average of ratings on most x similar items from this user. One key advantage of Item-based CF is the stability which is that the ratings on a given item will not change significantly over a given period of time. There are quite a few limitations for Collaborative Filtering, however. It does not handle sparsity well in a neighborhood, not is it it computationally efficient for larger number of users and products.

To tackle the sparsity problem, one can use Matrix Factorization [65], a method that decomposes the original sparse matrix to low-dimensional matrices with latent factors/features and less sparsity.

Let p_u and q_i denote the latent vector for user u and item i, respectively. Matrix Factorization estimates an interaction y_{ui} as the inner product of p_u and q_i

$$\hat{y}_u i = f(u, i|p_u, q_i) = p_u^T q_i = \sum_{k=1}^{K} p_{uk} q_{ik} \tag{4.4}$$

An intuitive explanation for why we need low-dimensional matrices is as follows: Suppose a user gives good ratings to movies Star Wars, The Matrix and Primer. This suggests that the user might like Sci-Fi movies and there may be many more Sci-Fi movies that this user would prefer. Latent features are expressed by higher-level attributes, and Sci-Fi category is one of the latent features in this case. What matrix factorization eventually gives us is how much a user is aligned with a set of latent features, and how much a movie fits into this set of latent features. The advantage of it over standard nearest neighborhood is that even though two users haven't rated

any same movies, it's still possible to find the similarity between them if they share the similar underlying tastes. More of Matrix Factorization will be discussed at a later chapter in this thesis.

4.2 Content-Based Filtering

A intuitive overview for content-based filtering is as follows

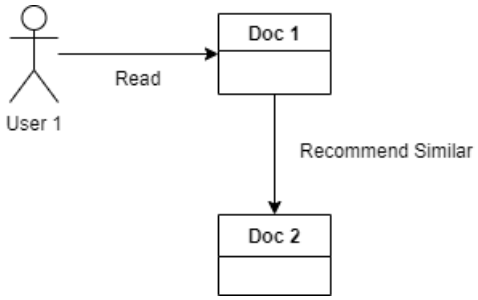

Figure 4.2 Content-based filtering.

The idea here indicates that if a user spent a considerable amount of time reading a document, he/she might be interested in documents of a similar nature. This can be extended to preferences for movies or music (by genre), or items in a shopping cart to name a few. Content based filtering is famous with text documents, articles and things of a similar nature.

A pseudo-code for recommending products or contents to the user in content based filtering could be the following -

Algorithm 3 Content-Based Filtering

1: Identify factors influencing the user's decision
2: Represent all items in terms of said factors
3: Make a tuple to store the strength of each contributing factor
4: Create user profiles and history based on the identifying factors and corresponding strengths
5: Make recommendations

This method shares the weakness of being computationally expensive and time consuming with the collaborative filtering method. In addition, unique identifying factors might need to be discovered and handpicked manually for a better accuracy which becomes infeasible for larger amounts of data.

It is for these reasons that a hybrid approach is often preferred which uses information from both user-item interactions and users/items' characteristics. The collaborative filtering information is included in a feature indicating whether an item is similar to the ones a user already prefers, and the content information includes whether the specific characteristics of the item match.

4.3 Personalized Recommendations

Model personalization is particularly useful for recommender systems. Because preferences between users differ a lot, incorporating additional user-specific context can boost performance [10] and by customizing the models to a small extent. A naive approach to personalization uses completely independent models which are trained for each user on his/her own device. For many reasons, this is not a preferred approach because it's time and resource intensive. In addition, there simply isn't enough data on every single user to train their own models well enough.

For Federated Learning systems, this comes naturally since we are already training on the user's computer. For the sake of privacy, the customized model is not shared with the server. Data leakage in the collaborative training can be controlled using differential privacy. To define the goals of personalization, consider the overall loss across all clients. If the client has customized the model, the loss is computed using that model. Otherwise, the global model is used. The loss should be smaller when using individually personalized models compared to using the central one and the overall quality of the global model should not deteriorate. New users will be able

to use this global model, this handling the cold start problem in machine learning and recommender systems [68].

Personalization is a new research area of Federated Learning that was proposed by [79]. MOCHA proposed by [97] can achieve personalization by learning separate but related models for each device while leveraging a shared representation via multi-task learning. This method can theoretically guarantee convergence but is limited in its ability to scale up to a larger network and is also limited to convex objectives. Another approach [22] is based on the star topology as a Bayesian network and performs variational inference during learning. This method is able to handle non-convex models but is also expensive to generalize to large federated networks. [109] explore transfer learning for personalization by running FedAvg after training a global model centrally on some shared proxy data.

4.4 Transfer Learning

One key problem which still remains is that of the *cold start problem* [68]. This is the case when a new user with no historical information needs to be given recommendations. The first family of methods in dealing with the cold start problem in recommender systems and machine learning is Transfer Learning, which is a common machine learning strategy for training models in situations where little data is available for the exact problem that needs to be solved [87] but a lot of data available by contrast for a more general version of the problem. For example, training a model to recognize a particular product in a shop's shelf, with its own SKU and other labels. If there are not many training images of this exact product available, it can be hard to fit a good model. However, a lot of training data exists for more general object detection tasks, such as the ImageNet dataset [25]. Transfer Learning tries to make use of the larger dataset to be able to solve the actual problem of fitting a model on the small dataset.

This is usually a two step process: first it is trained on a larger more generalized dataset, and in the second step, it is trained on the smaller and more specific dataset. This allows the training to start from a much better initial point. The intuition for this lies in how humans learn. When we learn to recognize a new object, we do not start from scratch but can learn more quickly because we already know how to recognize certain shapes and edges. Sometimes, only parts of the model are fine-tuned [86].

Transfer Learning has been used successfully in many parts of modern computer vision [86]. Natural language processing has also increasingly been making use of Transfer Learning. By first learning on a large corpus of general text, it becomes easier to work with inputs that represent words. Models for machine translation or text summarization are based on this idea. This idea can also be applied to Federated Learning. Training on the data of all users is the equivalent of training on the large, more generalized dataset here. This data is used as a good starting point for personalizing models locally.

As an analogue, Federated Learning is also a two step process where the global training takes place and then the individual users fine tune their models locally. Here, Transfer Learning techniques, such as freezing layers, are used. This technique can solve the general personalization problem, outlined in the previous section. Because the local training starts with an already well-trained model, it is easier to get a good custom model with little data and it also takes care of the cold start problem because new users start by using the central model, which was trained by a lot of users. However, the global model runs the risk of getting dated because as soon as users start fine-tuning their local model, collaborative training isn't possible anymore.

4.5 Learning to Rank Methods

An emerging topic that is useful for our purposes in this thesis is a technique from Informational Retrieval Systems called Learning to Rank (LTR) methods [74]. It is a technique for applying Supervised Learning to solve Ranking Problems [35]. The basic concept is that parameters are tuned automatically and the systems builds ranking models through the usage of hand labelled training data. It solves the ranking problem on a list of items and the aim of Learning to Rank is to devise an optimal ordering of those items, for example, search engine ranking.

It is usually achieved by posing it as a pairwise classification or regression problem, where one analyzes pairs of items at a time and decides the optimal ordering for that pair of items which is then used for the final ranking for all the results.

4.6 Deep Learning Based Methods

GPUs have made neural network based methods feasible today and as a result, a lot of emerging techniques have been centered around them. A comprehensive account of these can be found at [108]. The most famous recommender systems that use neural networks are [19] and [50].

4.6.1 Wide and Deep Learning for Recommender Systems

Wide and Deep Learning for Recommender Systems [19] leverages the usage of jointly trained wide linear models and deep neural networks to combine the benefits of memorization and generalization for recommender systems. Joint training of a Wide and Deep Model is done by backpropagating the gradients from the output to both the wide and deep part of the model simultaneously using mini-batch stochastic optimization. They tested this method by using the algorithm in [76], with different optimizers for the wide and deep parts of the network. The use case they chose was for recommending apps on Google's Play Store. Their choice of metrics was user

acquisition for the app that used the recommender system, a success was measured by whether the app was installed upon recommendation. Their training set consisted of 500 billion examples, and they use a warm-start method to avoid starting the process of batch training from scratch. Once the model is trained, they load it into the servers which have access to app candidates and user features which they use for ranking purposes and display it in the order of the corresponding ranks.

Their deep system showed an online acquisition gain of 2.9% with respect to the control and their wide and deep system showed an online acquisition gain of 3.9% which suggests that there are benefits to combining memorization and generalization in recommender systems.

4.6.2 Neural Collaborative Filtering

In [50] the idea was to replace the inner product on the latent features between the users and items with a neural architecture that could learn functions from the data. They use multi-layer perceptron to introduce non-linearities and learn the user-item interaction function. They do this by adopting a multi-layer representation to model the user-item interaction. Each layer of the network can be used to discover certain latent structure of the user-item interactions. The final output layer is the predicted score and training is performed by minimizing the loss between this score and its target value. It outperformed the state of the art [51] in 2016.

4.7 Federated Recommendation Systems

It has been approximated that about 1.7 megabytes of data is generated every second for each person on earth. Unsurprisingly, the amount of data points available today on each individual is astounding and preserving privacy is quickly becoming of paramount importance, especially in machine learning algorithms like recommender systems. Recommender systems are becoming increasingly ubiquitous and because

Federated Learning is inherently protective of privacy, using it for recommender systems potentially paves the way for their usage in domains previously unexplored due to data privacy concerns, like healthcare. In addition, it will also be able to leverage better and more relevant information to build more accurate models.

Building on the recommender systems we've already taken a look at in the previous sections, some work has been done on Federated Recommender Systems thanks to the ever increasing processing power and storage capacity on users' edge devices. One of the most important works and the current state-of-the-art in Federated Recommendation is [1], which can be briefly explained in steps as follows:

- The vectors of all the items are first updated on the global server and then distributed to each of the edge devices.
- Local training for user's item vectors is done on each device.
- Local gradients for each item vector is calculated. These local vectors are then back-propagated to the global server where the aggregation is done and the item vectors are updated

Some other work in this area is rather domain specific like this article by [110] which builds a privacy-preserving distributed online social recommender system with support for big data analytics. There is also content-based filtering in the form of a Federated Meta-Learning algorithm [17], which is a slightly different and a faster version of Federated Learning. In this paradigm, the model on each device is trained on a subsection of the training dataset to update the model parameters called a support set. The gradient is then computed on a different set called the query set. It's a complex method and a simplification in the form of an algorithm called REPTILE was proposed by [84] which instead of using separate support and query sets, and instead use a simple split validation that is sampled from a dataset of tasks. They

use the training set to train their algorithm, and using that they create an agent that has good average performance on the test set.

CHAPTER 5

PRIVACY IN RECOMMENDER SYSTEMS

As mentioned ealier in the thesis, differential privacy is utilized to make recommender systems privacy-preserving, and a differentially private recommender system that utilizes Alternating Least Squares and Stochastic Gradient Descent Solvers typically looks like Figure 5.1

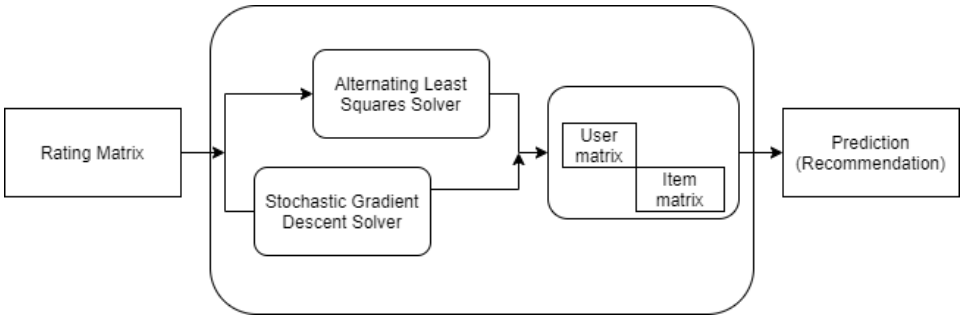

Figure 5.1 Differentially private recommender system

We've spoken about stochastic gradient descent earlier in the thesis. To introduce Alternate Least Squares (ALS), consider two variables X and Y. Alternate least squares fixes variable X and treats it as a constant and then the objective is a convex function of Y and then it fixes Y and optimizes X. This process is repeated until convergence. This is also the main solver for federated collaborative filtering which was given by [1].

Moving forward, a primary aim for decentralized algorithms is that they should be lossless in nature in comparison to their centralized counterparts and a hope for Federated Learning is that the data stays on the device with its contents being secure. However, the framework in Fig 5.2 relies on the users sending gradient information via plain text to the server and as such, for frameworks that communicate unencrypted gradients, it is possible to reverse-engineer sensitive individual rating data using

gradient information or even a part of the leaked gradient information, as we spoke about in an earlier discussion. Work has been done in the past by [36] who were able to reconstruct some of the faces that were used in the training of the model, and also by [2], [111] and [82] who all were able to successfully infer data about clients in some way based on model weights.

To this end, a solution is to use homomorphic encryption [40]. Applying homomorphic encryption to an machine learning model that relies on communication between two entities in the framework can be done according to the Fig. 5.2

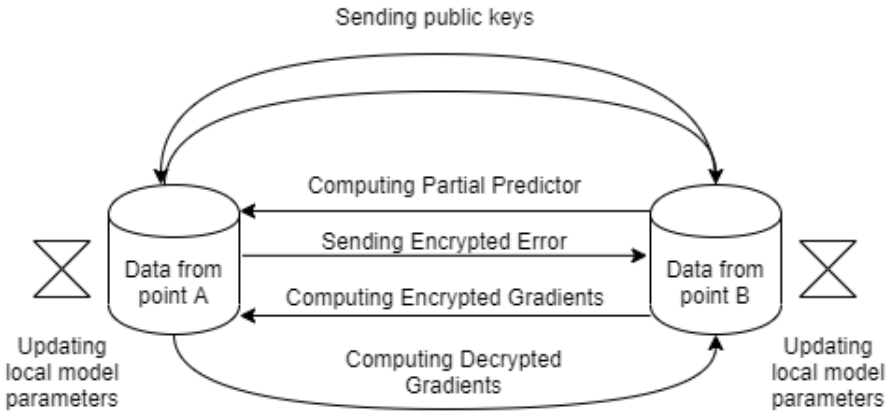

Figure 5.2 Model training using homomorphic encryption

It can be summarized as follows - public keys are exchanged between two parties, partial predictors are computed and the errors are encrypted and sent. These encrypted errors are used to calculate encrypted gradients. They are finally decrypted on the client side via the private key and used for updating the local model. The cycle then continues. It falls under the category of horizontal federated learning since it focuses on item-based federated recommendation and there is a large overlap of the items of the two rating matrices.

CHAPTER 6

IMPLEMENTATION METHODOLOGY AND RESULTS

6.1 Problem Statement

The objective of this thesis is to make a privacy-preserving federated recommender system that only utilizes parameters and gradients for training purposes in lieu of user data. With this, we address the problem of privacy in most existing recommender systems that perform training on data in a centralized manner. Additionally, we identify a potential problem in an existing federated recommender system framework which is that of gradient leakage whereby a malicious server might be able to infer sensitive user information based on even the gradients being sent from the devices [36], [2] [111]. We solve this by utilizing homomorphic encryption [40] to encrypt the gradients sent from user devices to the parameter server and perform computations on the server while keeping gradients encrypted. The hope is for the federated algorithms to have a more secure model even if it compromises slightly on certain prediction accuracy metrics.

6.2 Privacy Analysis

Availability of secure data is a very persistent problem in data science. One of the ways in which this challenge can potentially be overcome is if we are able to work with and answer questions using data that we cannot see. There are two approaches to solving this problem - to protect the data before it enters the model, which is differential privacy, or to build protection into the model, which is the Federated Learning approach. This section does a comparison of the two approaches.

6.2.1 Privacy from Differential Privacy Techniques

To reformulate an equation that was used in chapter 3; from a differential privacy standpoint, privacy is defined as follows - assume there is a dataset D_n. D_{n-1} represents the dataset D_n with one row removed. For a predicted value $f(D_n)$, we would like $f(D_{n-1})$ to be as similar to $f(D_n)$ as possible. In other words, if the probability distributions $P(f(D_n))$ and $P(f(D_{n-1}))$ were related by a factor e^{ϵ} such that

$$\frac{P(f(D_n))}{P(f(D_{n-1}))} \leq \epsilon \tag{6.1}$$

We would like ϵ to be as low as possible. In this scenario, perfect privacy would be defined as $\epsilon = 0$. There are many strategies to make the data private and the one I used here is to add random noise as it keeps the distribution roughly the same while keeping it relatively private. The accuracy has been plotted against different values for ϵ on the iris dataset and the results are as follows

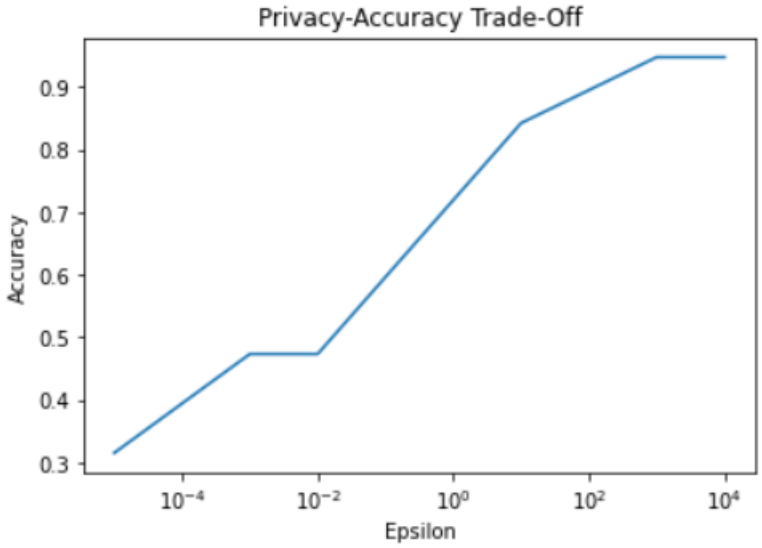

Figure 6.1 Privacy - accuracy tradeoff for differential privacy (trial 1)

Running the test again yields a different curve because of a different distribution of pseudo-random noise, but it still follows the pattern of accuracy of predictions getting better with an increasing ϵ value, suggesting that barring some occasional discrepancies attributed to the circumstantial nature of the trial, privacy would need to be compromised upon for increasingly better predictions.

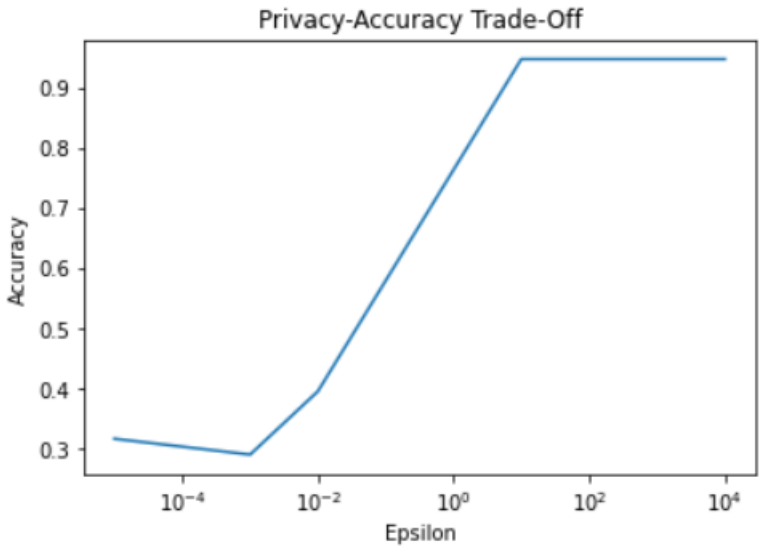

Figure 6.2 Privacy - accuracy tradeoff for differential privacy (trial 2)

There is a lot of work currently being done on Differential Privacy because of this, and the state-of-the-art in Differential Privacy focuses on combining it with other machine learning techniques to increase privacy and reduce user costs.

6.2.2 Privacy from Federated Learning Techniques

Federated Learning however does not try to change the data itself but instead focuses on training an algorithm across multiple different datasets without sharing information between the datasets themselves and sharing only training parameters with the server.

We use the OpenMind project PySyft (https://github.com/OpenMined/PySyft) that implements Federated Learning by creating toy version of the process. We create

two virtual workers and give them their own datasets for local training purposes. They will iterate through their datasets independently and separately and share the model weights with the global model which performs an averaging step in the server. After this, it trains until a predefined number of epochs is reached. Training is done using Stochastic Gradient Descent as the optimizer. Once the final global model has converged, the predictions from that model are then tested against new unseen local data that PySyft provides for the clients. We define a loss function using sum of squared errors which simply makes note of the difference between the predicted results and ground truth.

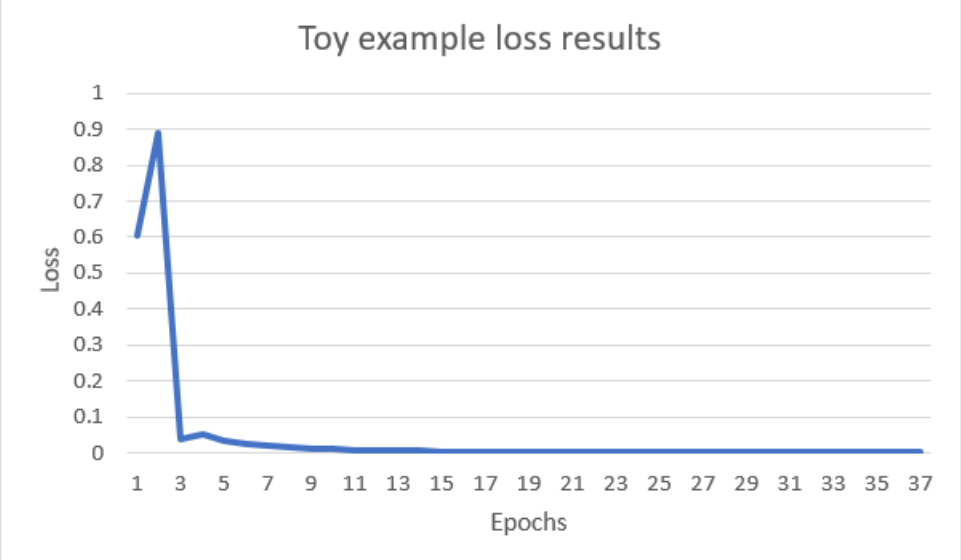

Figure 6.3 Loss performance for federated learning toy problem

Because this was implemented using a toy dataset set up in PySyft, we notice the workers start getting fairly accurate results over the course of only 3 epochs. While this is not going to be the case for any real world federated learning datasets, it demonstrates that this training paradigm is capable of convergence using training parameters instead of the data for training purposes. It also does not need for altering the dataset by adding noise to it, unlike Differential Privacy.

6.2.3 Gradient Leakage

Some papers have recently proven that gradient leakage occurs when gradient information in a client-server system is sent from the client to the server in plaintext. Low rank singular value decomposition like the one we perform can prevent the server from directly knowing raw preference data, but this is not sufficient. Various studies cited previously such as [111] and [2] show that we can obtain the private training set from the publicly shared gradients. The leaking only takes few gradient steps to process and is capable of obtaining the original training set. [82] also shows that gradient updates leak unintended information about participants' training data and develop passive and active inference attacks to exploit this leakage. This is an issue that we intend to address moving forward in the thesis by encrypting users' gradients and allowing the server to perform updates on the ciphertext instead of an averaged set of plaintext gradients.

6.3 Dataset

We use one of the benchmark MovieLens datasets which has a set of 100,836 ratings and 3683 tag applications made by 610 users across 9724 movies. It was collected in the late 1990s by the GroupLens research team at the University of Minnesota. We only take in consideration user who have rated at least 20 movies. User IDs are used for denoting the MovieLens users who were selected at random for inclusion with anonymized IDs. They are consistent between ratings.csv and tags.csv (i.e., the same id refers to the same user across the two files). Other fields include MovieID, timestamp, which represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970; and tags which are metadata collected about the movies.

To federate the dataset, it is split on a user-level with each user having their own datasets and preference information including ratings of the respective MovieIDs.

6.4 Data Preparation

The process of transforming the centralized unarranged dataset into a form which contains all the users with their respective datasets with their own train-test split is as follows:

- Initialize a dictionary of movieIDs, with the key as the ID and the value as the number of times it has been rated.
- Sort this in a descending order by the number of ratings.
- From this sorted list, select only the number of movies that will be used for training and testing purposes (to be pre-specified).
- Generated separate movie ID and user ID lists.
- Initialize a rating dictionary for the user ID list and associate each user ID with their rating details as their key respective value pairs.
- This forms their personalized dataset for each user ID which they will perform the train-test split on.
- Set a threshold for minimum number of ratings per user.
- Include the remaining users in the form of a sorted list.
- Perform a 70-30 split on each of the users' data.
- Update user ID list.
- There now exists a train-test split across each user who has rated a minimum number of movies.

Results of this data preparation phase which involved creating a train-test split for every user participating is as follows,

- Number of movies: 9724
- Number of users who rated at least 20 movies: 610
- Number of training ratings: 99006
- Number of test ratings: 1830

6.5 Baseline Assumptions

We make the following assumptions regarding the framework:

- The users are not adversarial in nature and do not provide malicious ratings which would severely affect performance. They have been assumed to maintain a certain level of consistency regarding their preferences which may or may not be practically applicable to the real world scenario.
- Federated learning has a lot of modules and requires a lot of independent devices to function. However, owing to the fact that the program is, in fact, running multiple instances of clients which are meant to functionally represent these multiple devices, work on communication efficiency techniques mentioned earlier in the background was out of the scope of this thesis. Communication instead takes place through function calls. In real life, this process would involve a lot more modules including web-sockets and network sockets.
- Federated averaging was performed using PySyft's existing API for the same and works to aggregate the gradients with a sufficient level of anonymity, including invoking normalization techniques for dealing with unusually large gradients that might expose a user.
- Secure Federated Aggregation using homomorphic encryption has a theoretical security guarantee [15] and we take their claim at face value.

6.6 Methodology Overview

As previously mentioned, the model will be conducting a federated collaborative filtering through matrix decomposition with the added security of homomorphic encryption. The intuition behind this is based on decentralized matrix factorization where each user profile is updated locally and the recommendation profiles of items are aggregated and updated by the server. The item recommendation gradients are encrypted by homomorphic encryption and the parameter server then aggregates these gradients.

We first implement the case of a regular collaborative filtering system using single value decomposition, which is a matrix factorization method. Collaborative filtering relies on user similarity and the baseline assumption being made is that users that are similar to each other in terms of movies they have previously rated, will be similar in the movies they will be rating in the future. This way, we're making filtering decisions for individual users based on the preferences and judgements of other users. If there are n users and m items, and each user rated a subset of m, $M = n$ x m would be the matrix denoting user-item rating pairs and $|M|$ would be the total number of ratings. User-profile and item-profile matrices are denoted by U and V respectively where U is the user profile matrix which consists of the left singular vectors with n users and d concepts for movie, giving it a dimension of n x d. V is the movie profile matrix consisting of the right singular vectors with m movies and d concepts, making it a m x d dimension matrix. d can be any number between 1 and the smaller number among n and m.

The movies and users are mapped into a 3D space where they are represented by points on it. The axes of this latent subspace are called factors. Through this mapping, we find low dimensional representations of users and movies such that the users that like these movies are closer to each other in this latent subspace.

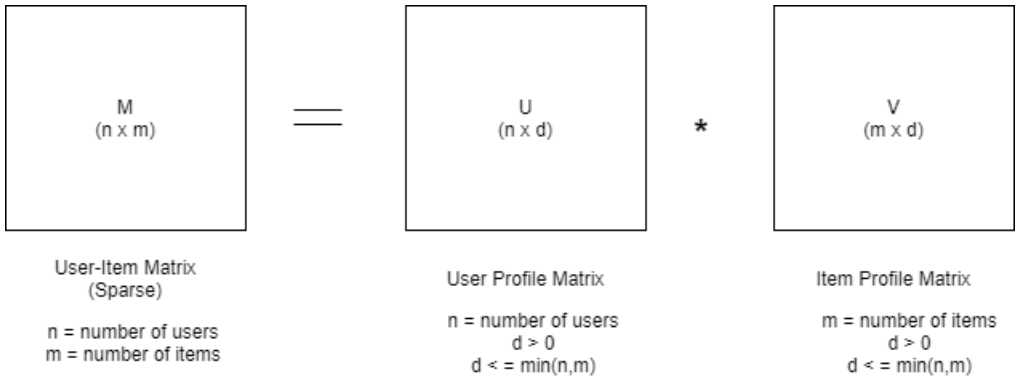

Figure 6.4 The matrix decomposition

Typically, a user only interacts with a small subset of the total number of movies, and the resulting matrix is therefore largely of a sparse nature. These matrices are used to predict user i's rating for item j, which is $< u_i, v_j >$. This can be posed as an optimization problem:

$$\min_{U,V} = \frac{1}{|M|}(r_{i,j} - < u_i, v_j >)^2 + \lambda||U||^2 + \mu||V||^2 \tag{6.2}$$

where λ and μ are small positive values used for regularization purposes.

We solve this optimization problem by minimizing the loss function using Stochastic Gradient Descent with the following update steps:

$$u_i^t = u_i^{t-1} - \eta * (-2 * v_j * (r_{i,j} - < u_i, v_j >) + 2 * \lambda u_i \tag{6.3}$$

$$v_j^t = v_j^{t-1} - \eta * (-2 * u_i * (r_{i,j} - < u_i, v_j >) + 2 * \lambda v_j \tag{6.4}$$

Then we build a federated scenario where rating information is stored on user's local devices and the model is trained on joint data and the iterative updating of gradients is split in two parts that are performed on the client side and server side. The stochastic gradient descent for user-profile matrix U, (6.3) is performed on each

user's device (the local update) while the stochastic gradient descent for item-profile matrix V, (6.4) is performed on the parameter server (the global update). The separation of these processes and the resulting decomposition of the matrix prevents the server from directly having raw data available to it. The diagram below shows how the federated collaborative filtering done through this matrix decomposition has been set up.

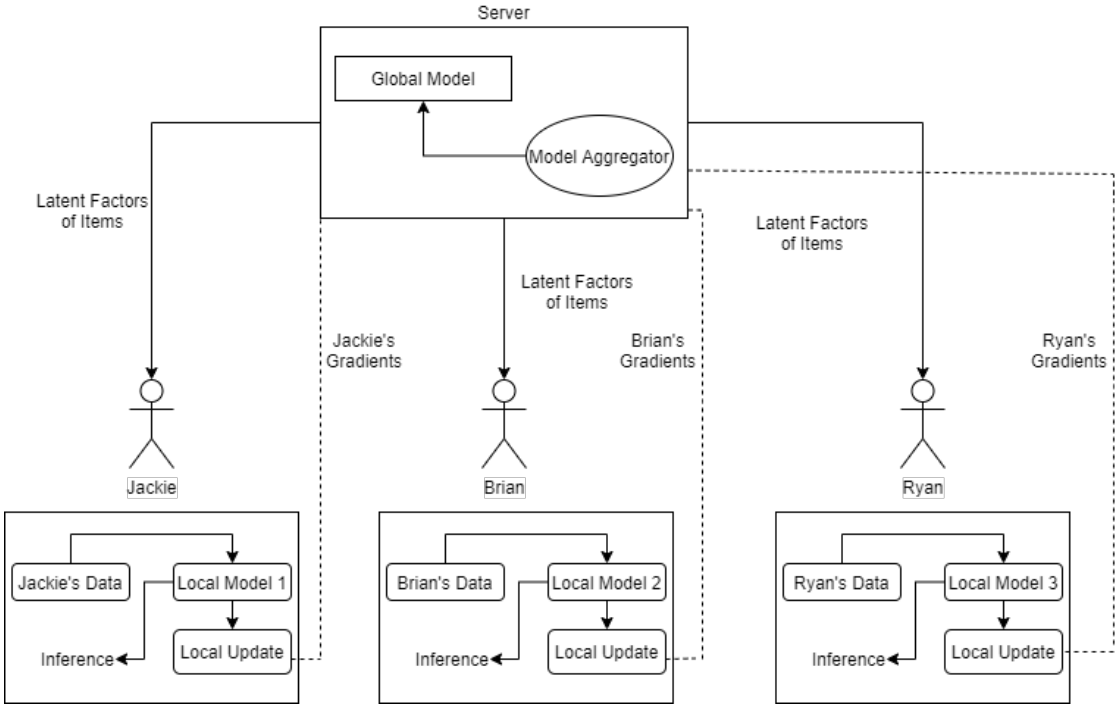

Figure 6.5 Federated collaborative filtering

The latent factors of items are sent from the server to the users to download, following which the users begin their local computation of gradients. These gradients are then sent back to the server for the next global updation of item factors.

We then enrich this algorithm with encryption whereby gradients are communicated safely in the manner defined by Fig 6.6 and operated on using partially homomorphic encryption. This addresses the problem of gradient leakage. To implement homomorphic encryption, we used Python's Paillier encryption protocol (https://github.com/data61/python-paillier) with their recommended library gmpy2

being used to accelerate the encryption process. The diagram for this combined process is as follows:

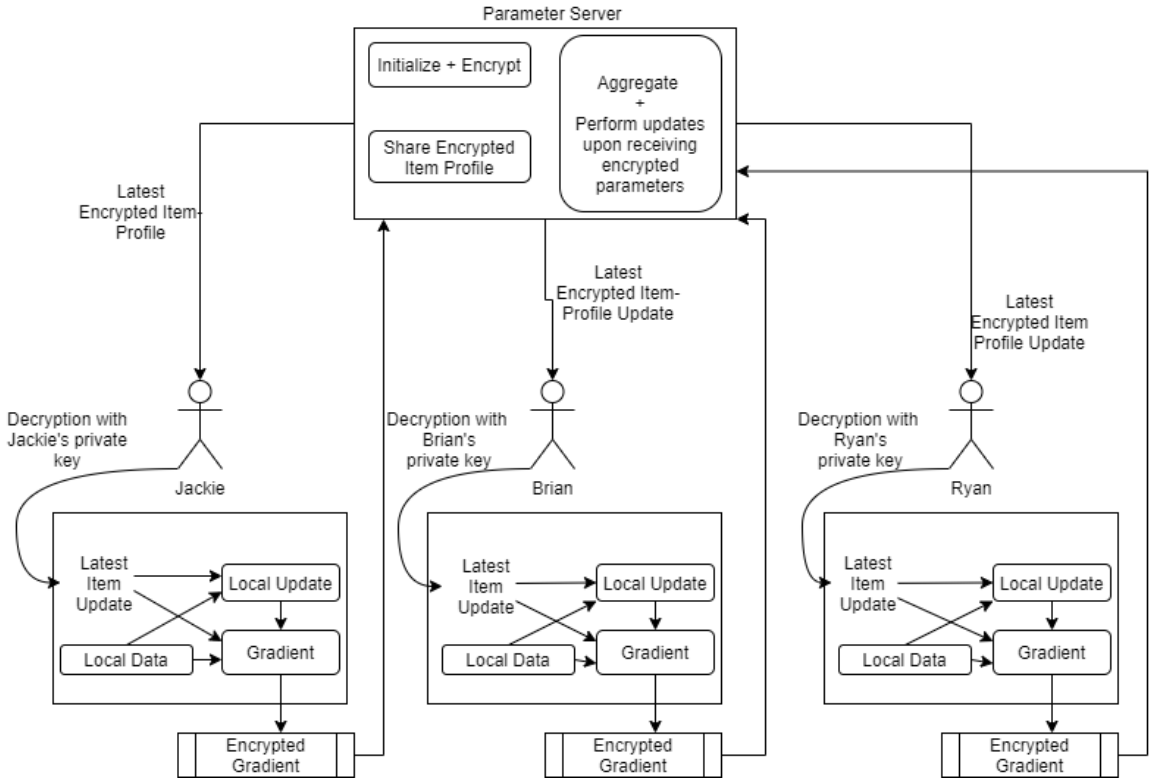

Figure 6.6 Federated collaborative filtering with encryption

6.7 Comparing the Gradient Update Calculations

Even though our aim is to utilize the same method for updating gradients across the frameworks, the implementations of the gradient update calculations are subject to their specific setups. The regular collaborative filtering version is done using the python library *surprise* (surpriselib.com) and the algorithm used was SVD, a matrix factorization method which solves the equations (6.4) and (6.3) on the same device. User and item factors are randomly initialized according to a normal distribution and has a learning rate of 0.005, with the regularization terms set to 0.02. These hyperparameters gave it the best RMSE values for that algorithm.

The federated collaborative filtering method solves the the equation (6.4) on the server and (6.3) on the user's edge device which is used for fine tuning with the user's data. The user profile is calculated for every single user and the gradients are averaged with the federated averaging algorithm on the server. The item profile is calculated once every epoch on the server after federated aggregation. The accuracy performance for this recommendation, and therefore the loss values varies every time because of its nature of randomized averaging of user gradients. Not directly training on the data evidently results in a slight loss of performance metrics compared to the performance of *surprise*'s implementation but is much safer as the preference data never leaves the user's device, which in this implementation has been represented by lists that contain training and testing data for every user. The item vector update is given as follows:

$$V_{ti} = V_{t-1i} - Gradient_i \tag{6.5}$$

where V_{ti} represents the item vector that will be downloaded by user i and $Gradient_i$ represents user i's gradient.

The federated collaborative filtering method with homomorphic encryption works much in the same way as the method stated previously above, with the additional feature of the gradients that get transmitted to the server being encrypted the server performing the item vector update on the ciphertext using the following:

$$C_{t+1V} = C_{tV} - C_G \tag{6.6}$$

where C_tV represents the ciphertext of the item vector and C_G represents the aggregated version of the ciphertext of the gradient received from the users. Thanks to the implementation of partially homomorphic encryption afforded by the python

library *python-paillier*, this does not result in a loss of performance, and the accuracy metric for this is very similar to the unencrypted version.

The computations for both the federated frameworks have a learning rate of 0.001 and regularization penalties of 0.01 as they produced the best results among the trials that were run.

6.8 Accuracy Analysis

An accuracy analysis was done based on results that were obtained from similar loss functions for a more legitimate comparison. The loss metric used to train the federated and secure federated model was a regularized least squares loss.

```
Out[129]: [Prediction(uid='373', iid='155', r_ui=4.0, est=3.0151309888935605, details={'was_impossible': False}),
           Prediction(uid='892', iid='31', r_ui=4.0, est=4.180904418166263, details={'was_impossible': False}),
           Prediction(uid='535', iid='389', r_ui=4.0, est=3.383708286715939, details={'was_impossible': False}),
           Prediction(uid='932', iid='1184', r_ui=3.0, est=3.183827816418472, details={'was_impossible': False}),
           Prediction(uid='919', iid='1173', r_ui=3.0, est=3.204160103716782, details={'was_impossible': False}),
           Prediction(uid='745', iid='177', r_ui=3.0, est=3.4986112054385385, details={'was_impossible': False}),
           Prediction(uid='157', iid='1016', r_ui=5.0, est=3.9689687216090763, details={'was_impossible': False}),
           Prediction(uid='758', iid='715', r_ui=4.0, est=3.914432515788601, details={'was_impossible': False}),
           Prediction(uid='907', iid='185', r_ui=4.0, est=5, details={'was_impossible': False}),
           Prediction(uid='807', iid='471', r_ui=4.0, est=3.937320524283284, details={'was_impossible': False}),
           Prediction(uid='805', iid='229', r_ui=2.0, est=2.707227738957559, details={'was_impossible': False}),
           Prediction(uid='109', iid='238', r_ui=2.0, est=4.102652953119813, details={'was_impossible': False}),
           Prediction(uid='312', iid='613', r_ui=5.0, est=4.493626826446467, details={'was_impossible': False}),
           Prediction(uid='910', iid='310', r_ui=3.0, est=3.3574212026323638, details={'was_impossible': False}),
           Prediction(uid='60', iid='56', r_ui=4.0, est=4.334204663380078, details={'was_impossible': False}),
```

Figure 6.7 Prediction results (centralized model)

```
loss 0.6272296920864687
Converged
rmse 1.1341257
Predictions on test data
[[3.8078952 4.1495767 2.9832554]
 [3.8350215 3.3668718 3.8331263]
 [2.9133773 1.8742661 2.3022234]
 ...
 [3.5285857 3.8559678 3.0152373]
 [3.2859359 2.851102  3.3164034]
 [3.4417262 3.493834  3.5696814]]
Real labels on test data
[[4.  4.  4. ]
 [5.  2.  4.5]
 [0.5 0.5 0.5]
 ...
 [5.  4.5 3. ]
 [3.  3.  4. ]
 [3.5 3.5 4. ]]
```

Figure 6.8 Prediction results (federated model)

The centralized collaborative filtering that is performed using SVD from the *surprise* Python library produces the best results, but comes at a cost of user privacy and training directly on user data. The best attainable root mean square value from the full user so far based on fine tuning the learning rate is presented below. The loss curve and RMSE value obtained from the model trained by the encrypted federated framework is almost identical to their unencrypted counterpart with a difference in value in the order of approximately $+-0.001$.

Table 6.1 Accuracy Analysis

RMSE	RMSE (from federated)	RMSE (from secure federated)
0.9455	1.13412	1.13534

If we observe the loss curves for the federated model with encryption and the federated model without encryption, it shows that the performance is nearly at par.

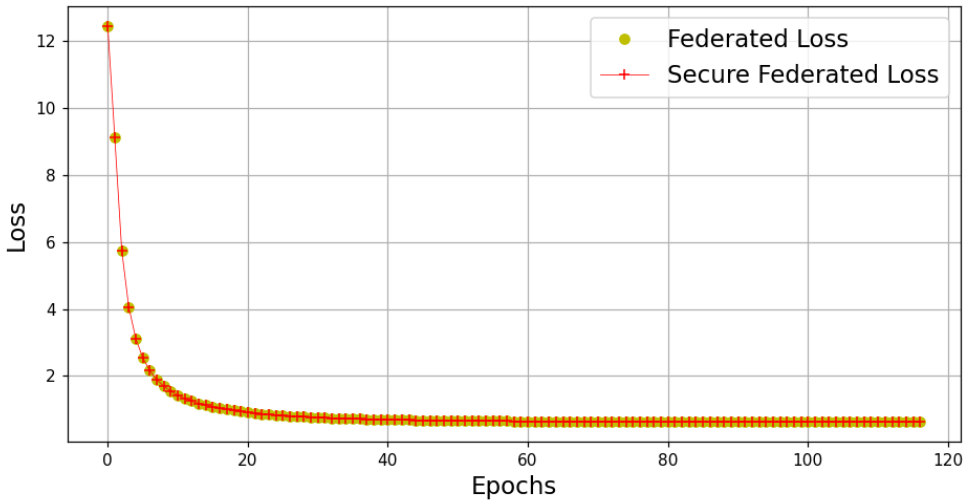

Figure 6.9 Loss performance for the unencrypted and encrypted models

This implies that the encryption does not affect algorithm performance with respect to outputs to a noticeable degree in terms of recommendation performance.

6.9 Computation Analysis

With the encrypted version of the federated recommender system framework, a caveat is the much larger training and testing time it takes because of the nature of calculations being performed by the system which are computationally and spatially expensive. However, we have found that the brunt of this computation is borne by the server and not the users, thus keeping the framework very much in line with the goals of federated learning which state that it respects the computational resources of the users.

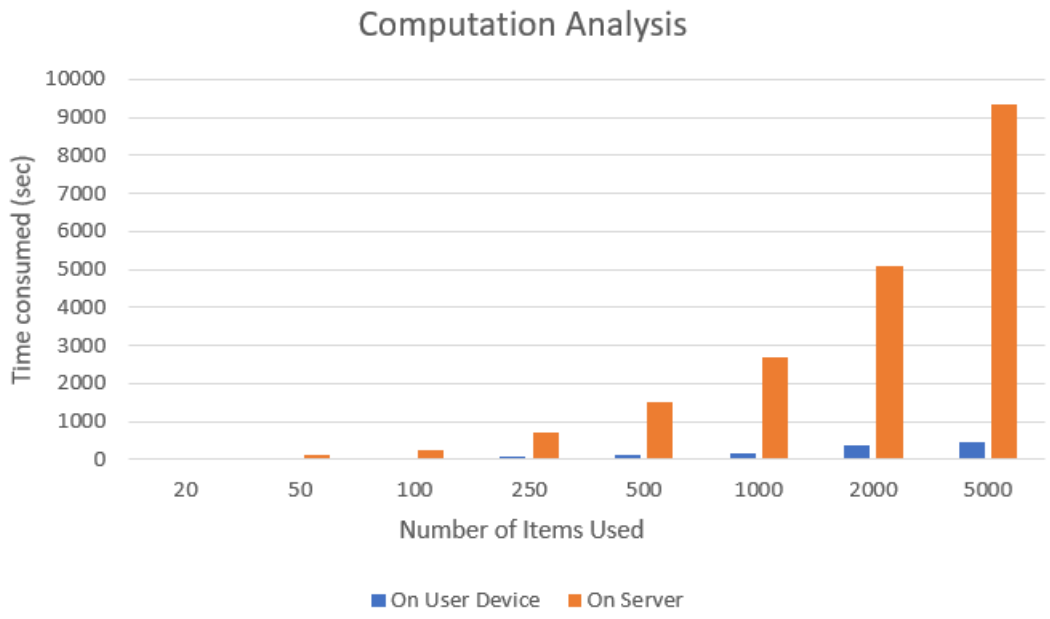

Figure 6.10 Analysis of computation time

6.10 Convergence Analysis

We start by analyzing the loss performance on a partial dataset, using number of items = 500 and changing the number of users doing the training. This shows that the performance of federated collaborative filtering algorithm increases greatly by increasing the number of users.

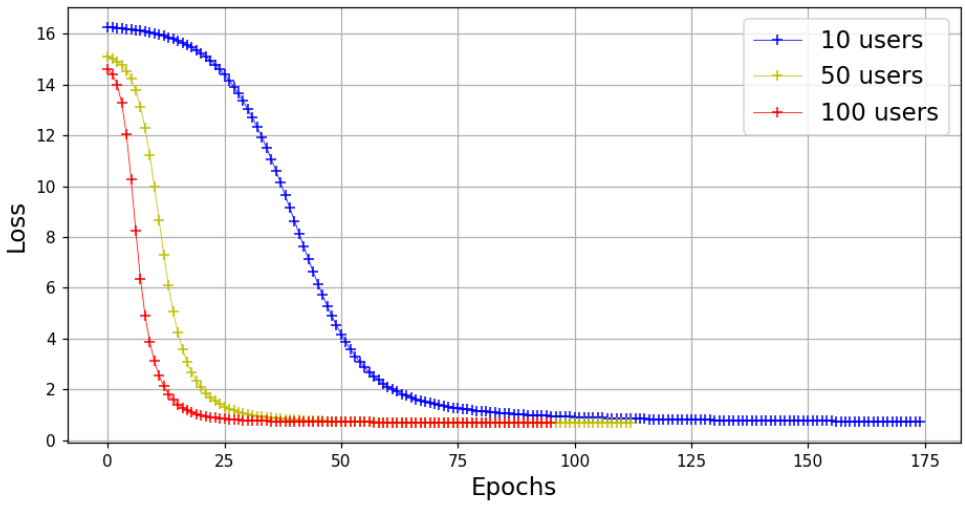

Figure 6.11 Loss comparison for 500 items

We then keep the number of users constant at a 100, and compare performance on the usage of 500 and 1000 movies to demonstrate that increasing just the number of items results in a better performance and a lower starting loss.

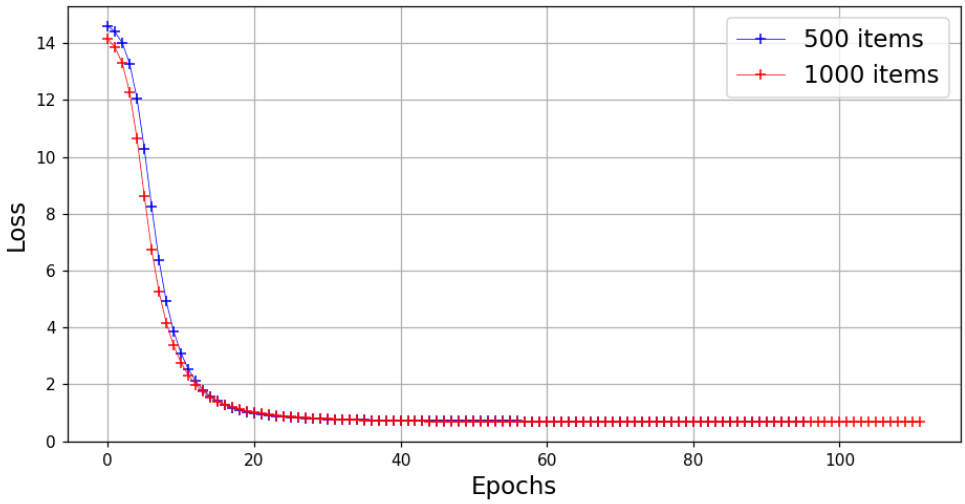

Figure 6.12 Loss comparison for 100 users vs variable items

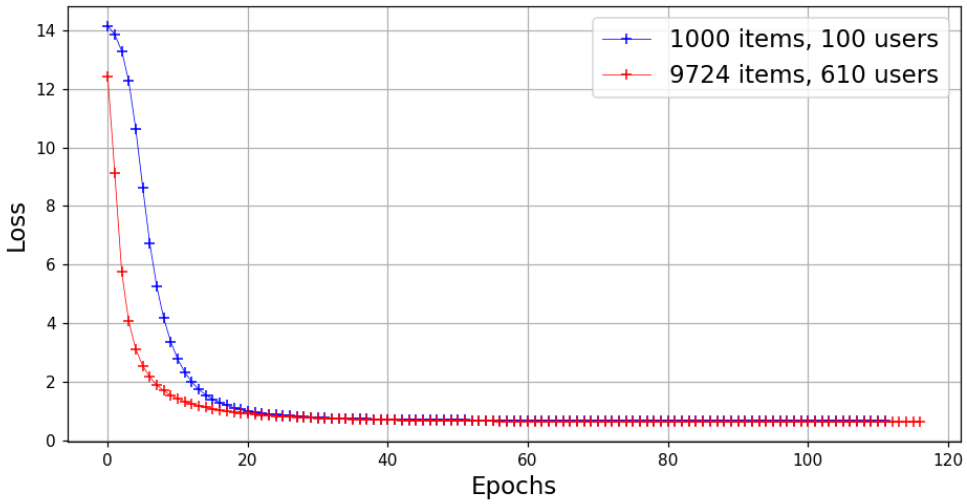

Figure 6.13 Subset vs full database

Finally, we plot the loss performance of a subset of the database with 100 users and 1000 items in the training and testing of the system, with the full database consisting of 610 users and 9724 movies (items). The loss curve starts at a much lower value and we notice it learning at a much faster pace per epoch as compared to training on the subset.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This thesis has been an interdisciplinary effort which explores Federated Learning, recommender Systems and cryptography. We started by giving an overview of Federated Learning and exploring the most pertinent works in the field as of early 2020, which is important to note because of the pace at which this very new domain is expanding. It then touches upon recommender systems and different cases of privacy that were the foundations of the novel work that was done here. We implement a case of collaborative filtering in the context of federated learning based on ideas set forth in [1], create a dataset that is conducive to federated learning, utilize matrix decomposition techniques to implement the algorithm then expand on it by taking cues from [2] and other researchers involved in differential privacy and cryptography who suggested that decomposition of matrices was not a sufficient means of hiding user gradients. This creates a new framework that suggests a possible solution to the problem of gradient leakage, which involved encrypting gradients and performing calculations on the encrypted gradients without them ever being decrypted by the server. Security of the aggregation protocol utilizing homomorphic encryption also happens to have [15]'s theoretical guarantee.

The results show that the federated version of the algorithm that uses encrypted gradients performs almost as well as its unencrypted counterpart, thus proving that partially homomorphic encryption is a viable tool that can be used to implement privacy in matrix decomposition based collaborative filtering methods without compromising much on accuracy as compared to its version that communicates gradients using plaintext to the server. The result suggests that given enough computational resources, it would perform just the same on the larger MovieLens

dataset consisting of 25 million entries. Because the algorithm does not pertain to specific dataset, as long as there is a representation of a sparse matrix M that can be decomposed into its constituent factors, it can be used for almost any dataset with the appropriate pre-processing steps and and the data can be split to train in a federated manner. An example of another such application would be a restaurant recommender based on [3]'s Yelp dataset using Natural Language Processing and techniques. It would take user inputs based on the kind of food they are looking for and run it through the algorithm that parsed through the reviews, using it as a corpus and finally outputting suggestions. It is easy to imagine this process being made much more accurate with the local updates and improvements being made on the user's device, much like what Google did with its implementation of mobile keyboard prediction which is currently being used in GBoard [48]. Similarly another potential use case is a better and a privacy-preserving music recommender system for streaming purposes which would be a more secure version of what Spotify or Pandora currently have.

An assumption that was made here was that the users participating in the recommendation are not malicious and that dishonest users in the data and will not attempt to train their own devices with bad recommendations or vote in a manner that will not help them. A case can be explored in the future where a certain level of dishonesty can be assumed to be present in the dataset and we can look into techniques for dealing with them to create a more robust version of this framework.

BIBLIOGRAPHY

[1] Muhammad Ammad-ud din, Elena Ivannikova, Suleiman A Khan, Were Oyomno, Qiang Fu, Kuan Eeik Tan, and Adrian Flanagan. Federated collaborative filtering for privacy-preserving personalized recommendation system. *arXiv preprint arXiv:1901.09888*, 2019.

[2] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

[3] Nabiha Asghar. Yelp dataset challenge: Review rating prediction. *arXiv preprint arXiv:1605.05362*, 2016.

[4] Shahriar Badsha, Xun Yi, Ibrahim Khalil, and Elisa Bertino. Privacy preserving user-based recommender system. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1074–1083. IEEE, 2017.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459*, 2018.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[8] Daniel Barth-Jones. The're-identification'of governor william weld's medical information: a critical re-examination of health data identification risks and privacy protections, then and now. *Then and Now (July 2012)*, 2012.

[9] Jeffrey S Beis and David G Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 1000–1006. IEEE, 1997.

[10] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction*, 18(3):245–286, 2008.

[11] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. *arXiv preprint arXiv:1811.12470*, 2018.

[12] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[13] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *CoRR*, abs/1902.01046, 2019.

[14] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1175–1191, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[16] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018.

[17] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. *arXiv preprint arXiv:1802.07876*, 2018.

[18] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *CoRR*, abs/1712.05526, 2017.

[19] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[20] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, and Qiang Yang. Secureboost: A lossless federated learning framework. *arXiv preprint arXiv:1901.08755*, 2019.

[21] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[22] Luca Corinzia and Joachim M Buhmann. Variational federated multi-task learning. *arXiv preprint arXiv:1906.06268*, 2019.

[23] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. 2004.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[26] David M Diez, Christopher D Barr, and Mine Cetinkaya-Rundel. *OpenIntro statistics*. OpenIntro, 2012.

[27] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[28] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[29] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[30] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L Lagendijk. Privacy enhanced recommender system. In *Thirty-first symposium on information theory in the Benelux*, pages 35–42, 2010.

[31] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L Lagendijk. Privacy-preserving content-based recommender system. In *Proceedings of the Fourth International Conference on Multimedia and Security*, pages 77–84. 2012.

[32] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE transactions on information forensics and security*, 7(3):1053–1066, 2012.

[33] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[34] Adrian Flanagan, Were Oyomno, Alexander Grigorievskiy, Kuan Eeik Tan, Suleiman A Khan, and Muhammad Ammad-Ud-Din. Federated multi-view matrix factorization for personalized recommendations. *arXiv preprint arXiv:2004.04256*, 2020.

[35] Lester R Ford Jr. Solution of a ranking problem from binary comparisons. *The American Mathematical Monthly*, 64(8P2):28–33, 1957.

[36] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[37] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *CoRR*, abs/1808.04866, 2018.

[38] Dashan Gao, Yang Liu, Anbu Huang, Ce Ju, Han Yu, and Qiang Yang. Privacy-preserving heterogeneous federated transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2552–2559. IEEE, 2019.

[39] David G Garson. Interpreting neural network connection weights. 1991.

[40] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[41] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *CoRR*, abs/1712.07557, 2017.

[42] Jack Goetz, Kshitiz Malik, Duc Bui, Seungwhan Moon, Honglei Liu, and Anuj Kumar. Active federated learning. *arXiv preprint arXiv:1909.12641*, 2019.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[44] Andreas Griewank. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, pages 389–400, 2012.

[45] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[46] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[47] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[48] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[49] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[50] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[51] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558, 2016.

[52] James Hendler. Avoiding another ai winter. *IEEE Intelligent Systems*, (2):2–4, 2008.

[53] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.

[54] Amir Jalalirad, Marco Scavuzzo, Catalin Capota, and Michael Sprague. A simple and efficient federated recommender system. In *Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, pages 53–58, 2019.

[55] Mathias Johanson, Stanislav Belenki, Jonas Jalminger, Magnus Fant, and Mats Gjertz. Big automotive data: Leveraging large volumes of data for knowledge-driven product development. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 736–741. IEEE, 2014.

[56] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[57] Maja Kabiljo and Aleksandar Ilic. Recommending items to more than a billion people. *Retrieved May*, 2:2018, 2015.

[58] Harmanjeet Kaur, Neeraj Kumar, and Shalini Batra. An efficient multi-party scheme for privacy preserving collaborative filtering for healthcare recommender system. *Future Generation Computer Systems*, 86:297–307, 2018.

[59] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Better communication complexity for local sgd. *arXiv preprint arXiv:1909.04746*, 2019.

[60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[61] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org, 2017.

[62] Jakub Konecný, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *CoRR*, abs/1511.03575, 2015.

[63] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[64] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

[65] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[67] Miklós Kurucz, András A Benczúr, and Károly Csalogány. Methods for large scale svd with missing values. In *Proceedings of KDD cup and workshop*, volume 12, pages 31–38. Citeseer, 2007.

[68] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pages 208–211, 2008.

[69] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[70] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[71] Paul Pu Liang, Terrance Liu, Liu Ziyin, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*, 2020.

[72] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

[73] Boyi Liu, Lujia Wang, Ming Liu, and Chengzhong Xu. Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. *CoRR*, abs/1901.06455, 2019.

[74] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[75] WANG Luping, WANG Wei, and LI Bo. Cmfl: Mitigating communication overhead for federated learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 954–964. IEEE, 2019.

[76] H Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. 2011.

[77] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[78] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. *CoRR*, abs/1710.06963, 2017.

[79] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. *arXiv preprint arXiv:1710.06963*, 2017.

[80] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[81] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with d-operand multiplications. In *Annual Cryptology Conference*, pages 138–154. Springer, 2010.

[82] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[83] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *arXiv preprint cs/0610105*, 2006.

[84] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[85] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. A performance evaluation of federated learning algorithms. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, DIDL '18, page 1–8, New York, NY, USA, 2018. Association for Computing Machinery.

[86] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[87] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[88] Daniel Peterson, Pallika Kanani, and Virendra J Marathe. Private federated learning with domain adaptation. *arXiv preprint arXiv:1912.06733*, 2019.

[89] Foster J Provost and Daniel N Hennessy. Scaling up: Distributed machine learning with cooperation. In *AAAI/IAAI, Vol. 1*, pages 74–79. Citeseer, 1996.

[90] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[91] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[92] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. Stealthy poisoning attacks on pca-based anomaly detectors. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):73–74, 2009.

[93] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[94] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *International Conference on High Performance Computing for Computational Science*, pages 1–25. Springer, 2010.

[95] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014.

[96] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4424–4434. Curran Associates, Inc., 2017.

[97] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[98] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.

[99] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, AISec'19, page 1–11, New York, NY, USA, 2019. Association for Computing Machinery.

[100] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

[101] Jianyu Wang, Anit Kumar Sahu, Zhouyi Yang, Gauri Joshi, and Soummya Kar. Matcha: Speeding up decentralized sgd via matching decomposition sampling. *arXiv preprint arXiv:1905.09435*, 2019.

[102] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2512–2520. IEEE, 2019.

[103] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[104] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[105] Xin Yao, Tianchi Huang, Rui-Xiao Zhang, Ruiyu Li, and Lifeng Sun. Federated learning with unbiased gradient aggregation and controllable meta updating. *arXiv preprint arXiv:1910.08234*, 2019.

[106] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[107] Justin Zhan, Chia-Lung Hsieh, I-Cheng Wang, Tsan-Sheng Hsu, Churn-Jung Liau, and Da-Wei Wang. Privacy-preserving collaborative recommender systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(4):472–476, 2010.

[108] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1), February 2019.

[109] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[110] P. Zhou, K. Wang, L. Guo, S. Gong, and B. Zheng. A privacy-preserving distributed contextual federated online learning framework with big data support in social recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2019.

[111] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14747–14756, 2019.

[112] Hankz Hankui Zhuo, Wenfeng Feng, Qian Xu, Qiang Yang, and Yufeng Lin. Federated reinforcement learning. *CoRR*, abs/1901.08277, 2019.