DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2019

Differentially Private Federated Learning

NIKOLAOS TATARAKIS

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Differentially Private Federated Learning

Nikolaos Tatarakis

October 2019

Master in Machine Learning
PI at EPFL: Boi Faltings
Supervisor at EPFL: Aleksei Triastcyn
Supervisor at KTH: Mats Nordahl
Examiner at KTH: Elena Troubitsyna
School of Electrical Engineering and Computer Science

Abstract

Federated Learning is a way of training neural network models in a decentralized manner; It utilizes several participating devices (that hold the same model architecture) to learn, independently, a model on their local data partition. These local models are then aggregated (in parameter domain), achieving equivalent performance as if the model was trained centrally. On the other hand, Differential Privacy is a well-established notion of data privacy preservation that can provide formal privacy guarantees based on rigorous mathematical and statistical properties. The majority of the current literature, at the intersection of these two fields, only considers privacy from a client's point of view (i.e., the presence or absence of a client during decentralized training should not affect the distribution over the parameters of the final (central) model). However, it disregards privacy at a single (training) data-point level (i.e., if an adversary has partial, or even full access to the remaining training data-points, they should be severely limited in inferring sensitive information about that single data-point, as long as it is bounded by a differential privacy guarantee). In this thesis, we propose a method for end-to-end privacy guarantees with minimal loss of utility. We show, both empirically and theoretically, that privacy bounds at a data-point level can be achieved within the proposed framework. As a consequence of this, satisfactory client-level privacy bounds can be realized without making the system noisier overall, while obtaining state-of-the-art results.

Sammanfattning

Federated Learning är ett sätt att träna neurala nätverksmodeller på ett decentraliserat sätt. Metoden använder flera deltagande enheter med samma modellarkitektur för att självständigt lära sig en modell på sin egen lokala datapartition. Dessa lokala modeller aggregeras sedan i parameterdomänen och uppnår motsvarande prestanda som om modellen tränats centralt. Å andra sidan är Differential Privacy ett väletablerat begrepp inom integritetsskydd som kan ge formella integritetsgarantier baserade på rigorösa matematiska och statistiska egenskaper. Majoriteten av den aktuella litteraturen, i korsningen mellan dessa två områden, beaktar bara integritet ur en klients synvinkel. Närvaron eller frånvaron av en klient under decentraliserad träning bör inte påverka fördelningen över parametrarna för den slutliga, centrala, modellen. Emellertid bortser man från sekretess på nivån för enskilda tränings-datapunkter, dvs. om en angripare har delvis eller till och med full tillgång till de återstående träningsdatapunkterna, bör dess möjlighet att dra slutsatser om känslig information rörande den enskilda datapunkten vara starkt begränsad, förutsatt att den omfattas av en differentierad integritetsgaranti. I det här examensarbetet föreslår vi en metod för integritetsgarantier med minimal förlust av användbarhet. Vi visar, både empiriskt och teoretiskt, att integritetsgränser på en datapunkt kan uppnås inom det föreslagna ramverket. Som en följd av detta kan tillfredsställande sekretessgränser på klientnivå realiseras utan att göra systemet brusigare på en övergripande nivå, samtidigt som man uppnår bästa möjliga resultat.

Contents

Chapter 1

Introduction

The ubiquity of the internet and social media, personal computers, handheld devices, such as mobile phones, tablets, wearables, and so on, has given rise to an enormous stream of data. To put this into perspective, about 2.5 quintillion bytes of data, are generated per day, according to the study in [45].

There is an increased interest in manipulating all this data volume to create models, based on experience, that can help us to extract insights and ease people's everyday lives. Most of the current systems gather all this available data and process it centrally to create (or learn) a model. Then, given the learnt model, systems can answer related queries (inference).

Examples of such systems we use on a daily basis, include the text auto-complete on our mobile devices, movie and music recommendations, personal assistants, email spam filtering, and more. The majority of these services require access to every kind of personal data in order to be as accurate as possible. This has led to a growing concern about how machine learning intrudes people's privacy.

Providing access to private data suggests that organizations (e.g., companies, research institutes) involved in the processing of this data should be regarded as *trusted curators*[1], and at the same time, their data infrastructure is supposed to be safe enough to prevent any kind of data leaks. In reality, there are many examples where neither the scientists nor the infrastructure proved to be trusted or safe enough to protect people's personal information [4, 63, 75, 97].

Recently, a great amount of effort has been made to address these kinds of issues within machine learning domain. Differential Privacy [24], which is a relatively new concept, and mainly used to protect the privacy of statistical databases, has attracted quite a lot of attention by the machine learning community as it can promise an enhanced level of privacy even if the trained model is released to the public, deliberately or accidentally.

[1]By trusted curator, we refer to a trusted data collector who analyzes the users' private data and adds noise to it in an elaborate way, such that both differential privacy and high utility (e.g., high classification accuracy) can be realised.

1.1 Specified Problem Definition

A simplistic approach to create machine learning models based on users' data is by allowing users to share their data with a central server. Then, the server applies the learning algorithm once a considerable amount of data has been gathered. Although this method can be quite effective both for learning and for inference, it has two clear disadvantages. Firstly it requires massive amounts of centralized computational power. Secondly, it disregards the privacy of the participating entities.

Federated Learning [65], is a recent advancement in Machine Learning which allows the learning procedure to be decentralized. That is, a mobile device should be able to run a typical Stochastic Gradient Descent algorithm (SGD) locally on its data. This provides a significant advantage to a user's privacy since her data is not collected and processed centrally.

Federated Learning by itself is not enough. As there is still a need for an aggregated model that can answer to the queries, it means that (aggregated) gradients can still reveal information about the actual data [31, 69]. In this case, Differential Privacy (DP) can provide the needed guarantees and well-defined bounds, using a rigorous statistical framework, so that the aggregated model cannot "leak" any sensitive information of the participating entities.

In order to keep a model private under differential privacy framework, noise must be either injected into the gradients (during the training procedure) or to the final learnt model parameters (after the training procedure has been concluded). This noise, if applied over many training rounds, can be the most prominent reason for accuracy deterioration. Thus, there is a major trade-off to consider here: The more noise you inject, the more private the solution becomes, however, less accurate. That being said, developing differentially private machine learning models in a non-trivial task.

Research Question

Despite the recent advancements in research, where Machine Learning and Differential Privacy intersect, there is another challenge to be considered within the current privacy framework; The majority of the proposed solutions (see Chapter 2), consider the noise addition centrally (that is from the server-side). This is a rather convenient way to add noise, primarily for two reasons; 1) Noise is only added every time the models are being aggregated, thus these methods result in fewer noise injections & 2) They consider that the central aggregator is always a trusted entity (trusted curator).

Although injecting less noise is attractive, since the utility increases, one has to rely on a trusted curator for the final model, which, strictly speaking, poses a significant liability risk to involved users' privacy. To this end, we would like to know, *how* creating *locally* differentially private models can minimize this risk as their aggregation can additionally achieve *global* privacy guarantees. In other words, we would like to investigate whether moving the Privacy Mechanism to the clients' side can address this issue and to what extent is this feasible (e.g.,

do we have to sacrifice the model's utility for this extra layer of privacy?). The importance of this study is that end-to-end privacy guarantees can be formally shown, and thus reduce the *trusted* curator into a simple curator who just aggregates data.

We study this from a theoretical point of view, showing how local [2] privacy can provide, formal, global privacy guarantees, and empirically by implementing differentially private models and comparing their accuracy and privacy guarantees to the current state-of-the-art.

The Question: *Can we provide **both** data-level privacy guarantees and client-level privacy guarantees by moving the Privacy Mechanism into the clients, so that we don't need a **"trusted"** curator for the data aggregation?*

1.2 Contributions

The principal contributions brought by this thesis can be summarized as follows:

- We propose and evaluate a new framework for differentially private machine learning within federated learning paradigm.
- We show empirically and theoretically that both local and global privacy guarantees hold within the proposed framework.
- We illustrate quantitative results of this study with regard to the trained models' accuracy and their corresponding privacy guarantees achieved, comparing them with relevant work in the field.

1.3 Ethical & Societal Aspects

Lately, data privacy & data protection has been very high on the agenda of both governments and other (private) institutions. For example, with the General Data Protection Regulation (GDPR), the European Union legally bounds governments and institutions to protect their users' data, which basically prevents them from sharing it with the outside world. Although this is an initiative towards the correct direction for privacy (at least from an ethical and moral point of view), a certain consequence arises.

To put this into perspective, under this directive, a University Hospital with a lot of tumor images (that can potentially be used in cancer prediction), is not allowed to share this data publicly as it constitutes a privacy breach. This is a very restrictive consequence as scientists' work is severely hindered. In fact, scientists are either highly limited or even not allowed to access this data volume to develop more sophisticated and accurate predictive models.

This is a real problem that holds society back, not only for understanding and

[2] Not to be confused with "Local (Differential) Privacy" which is a slightly different variation of Differential Privacy [50]. Here, by local, we refer to each clients' *local* DP model.

curing diseases but also for implementing machine learning tools for understanding society itself. Therefore, developing state-of-the-art, privacy-preserving mechanisms is of vital importance to tackle these kinds of problems and let both society and science advance hand-in-hand.

1.4 Sustainability

Training highly accurate Deep Learning models on enormous datasets requires considerable amounts of computational power. As shown in [94], training complex models with a rather large number of free parameters can take up to several days to compute, and it can produce up to 626,000 lbs of CO_2 emissions. In comparison, the average human life, during a 1 year time period, produces approximately 11,000 lbs of CO_2 emissions according to the same source.

Training neural networks under Federated Learning may be more energy efficient. In [11], authors argue that mobile applications (such as "on device item ranking"), if they are computed on users' devices instead of a server, expensive calls (to the server) including bandwidth, latency and power consumption can be eliminated.

In [30], Fehske et al. estimate that *operating* a smartphone during a year it consumes about 7kWh/year, which translates to roughly 4.5 lbs of CO_2 emissions per year (EU average). Hopefully, with careful planning and smarter models that take advantage of the Federated Learning framework, we could perhaps keep the mobile emissions to those levels without increasing it further.

1.5 External Supervision

The present work has been carried out in its entirety at the Swiss Federal Institute of Technology, Lausanne (EPFL). In particular, the project has been conducted in collaboration with the Laboratory for Artificial Intelligence (LIA), which focuses on Artificial Intelligence (AI) and strives to push the research boundaries further in AI and related fields. This work has been completed under the mentorship of Aleksei Triastcyn, whom I would like to cordially thank for supervising and assigning me with this interesting project.

1.6 Organization of the Thesis

This Thesis is organised as follows; Formal definitions and related work follows in Chapter 2, Background. The proposed framework is introduced in Chapter 3. Experimental results follow in Chapter 4, and, finally, we draw our conclusions in Chapter 5.

Chapter 2

Background

2.1 Motivation

In 2007, Netflix held a competition for the best collaborative filtering algorithm that can predict a user's rating on a film based on previous users' ratings. All user information in the datasets, such as names (even movie names), were replaced by numbers specifically assigned for competition. In [75], authors showed how they managed to reveal almost 99% of the sensitive data of the aforementioned dataset (including movie names and user names) just by using auxiliary public information from Internet Movie Database (IMDB).

Anonymizing the data is, in fact, a rather weak approach to preserve the privacy of an individual. In 2006, AOL released anonymized search logs of roughly 600K randomly selected users, to be used for research purposes. The anonymization was insufficient because the logs contained personal details of the users. Barbaro et al. in [63] show how users' identities can be compromised, for example, by cross-referencing these details with phone-book listings. Another notable case was the de-anonymization of medical records that occurred in 1997; In [4], Barth-Jones discusses how attackers managed to re-identify the medical records of the Massachusetts Governor. Attackers used multiple datasets of anonymized medical records, which they basically matched with *publicly* available voter registration records. More recently, another study [97] shows how Genome Data can be linked to public records to identify participants of the Personal Genome Project.

From a Machine Learning perspective, given a trained model, we would like to build a framework that highly limits the potential adversary from recovering sensitive information from the dataset by analyzing the trained model (model's parameters), even if they possess auxiliary information. That is, when dealing with private and sensitive data sets (for example images for cancer prediction), we would like to minimize the risk that potentially memorized data patterns (by the trained model's parameters) can be linked to other publicly available information and therefore leak personal information about the people to whom this data belong.

2.1.1 Attacks in Machine Learning

Machine Learning has been recently at the epicenter of many different types of attacks. We could cluster them into two main categories: Security and Privacy attacks.

In the former category, there are Adversarial Machine Learning [57] related attacks that are mainly focused around creating fake data examples (e.g., images that look like class A but created in such way to maximize the probability of being classified as B [36, 56, 98]) with the purpose to fool machine learning models. This type of attacks can be further categorized into Evasion and Poisoning. In Evasion [8], the adversary's goal is, at test time, to classify positive samples as negative, for example, by perturbing the test inputs. On the other hand, Poisoning attacks aim to contaminate the training set with adversarial examples (e.g., through label flipping [9]). Poisoning approaches eventually compromise the learning process such that the trained model will misbehave for certain test inputs. Defence mechanisms have been suggested in [109, 114, 91] for Evasion attacks and in [47, 93] for Poisoning attacks.

Privacy Attacks in Machine Learning

In the Privacy domain, which is the focus of this thesis, the adversary's goal is to extract sensitive, private information about the underlying training set of the model, or the model itself. In this case, there are two main attack techniques, namely, Membership Inference and Model Inversion.

In Membership Inference, as the name implies, the adversary wants to know, for example, whether a data-point he holds is part of the training set [89, 111, 81, 102]. In many environments, such as predictive medicine, this can pose a considerable privacy threat. For example, suppose that an online health service provider suggests disease predictions based on genotype data. In addition, suppose that an adversary (e.g., an insurance company) holds genotype data of some prospective customer A. If they could infer that A was part of that training set (i.e., they are aware that A had a certain disease), the company could charge much more, or, even deny an insurance policy to A for some other, however, related disease. Membership Inference attacks have also been explored in the context of generative models [39], where authors are basically using GANs to exploit overfitting generative models to detect samples that were part of the training set.

On the other hand, Model Inversion attacks target the learnt model itself. They aim to reconstruct an aggregated internal representation of the model, such as a class representation [31, 13]. For example, if we consider a face recognition model, it is possible under model inversion to reconstruct an image that maximizes the probability of being from some particular class (i.e., an actual person in this example). Although the reconstruction might not be very realistic, it can still reveal the identity of the targeted person (e.g., in [31]). Defence mechanisms that address such issues (Membership Inference and Model Inversion) are principally based on the notion of Differential Privacy, for example, [101, 1, 88].

2.2 Machine Learning

Conceptually speaking, Machine Learning (ML) strives to find all these ways to take advantage of the available data and provide insights out of it, without trying to explicitly define the relationships between the data (i.e., without using explicit instructions). Rather "abstractly", ML tries to reach a conclusion in a similar fashion to the way humans process information, using some *prior* knowledge or experience. That is, ML algorithms can "learn" from representative examples of input (and output) data.

A more structured definition by [72] denotes that "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". Applications of ML algorithms include but are not limited to learning how to recognize visual object categories [55], learning how to translate linguistic data [95], learning how to play GO [90] or, even learning how to suggest our favorite music [103] and so on and so forth.

We can map ML algorithms in the following broad categories; Supervised Learning (e.g., Classification, Regression), Unsupervised Learning (e.g., Clustering, Dimensionality Reduction), Semi-Supervised Learning & Reinforcement Learning. In this thesis, we will deal mostly with Differential Privacy within Supervised Learning. Therefore, a succinct introduction will be provided in this area. For an in-depth and comprehensive reading in ML literature, we highly suggest [35, 10, 73, 32, 96].

A Probabilistic Perspective

In supervised learning, we suppose that there is an "input" space $X \in \mathbb{R}^p$ and an "output" space Y. Depending on the supervised learning task, Y can be either real valued or categorical. The training data S, consists of n samples that we suppose that they are drawn independently and identically distributed (i.i.d.) from a probability distribution $\mu(z)$ on $Z = X \cdot Y : (x_1, y_1), ..., (x_n, y_n)$, that is, $z_1, ..., z_n$. Therefore, we are looking for a function $f(X)$ that is able to predict a target Y given the (new) input values X, that is $y_{pred} = f_S(x_{new})$. We use the conditional probability of y given x, (i.e., $p(y|x)$) to model this relationship as follows:

$$p(x, y) = p(y|x) \cdot p(x). \tag{2.1}$$

If y is real valued, then we say that we have a regression problem and so we are looking for a mapping such as $f : \mathbb{R}^p \mapsto \mathbb{R}$. If the output space is a categorical value of y in range of k, the mapping we are looking for is $f : \mathbb{R}^p \mapsto \{1, ..., k\}$, and hence we call it Classification.

Loss Function

Whether the problem is Classification or Regression, it is clear that there exists a hypothesis space $\mathcal{H}$, which is the space of functions that our learning algorithm is allowed to search in. This is also known as function approximation, as we are basically trying to find the function f out of a set of possible functions $F \in \mathcal{H}$ that best maps inputs to outputs. Therefore, we need a way to quantify

"how good", or "how bad" a function f is for the task.

We can now introduce the Loss function L: Given some particular pair of inputs x and outputs y the function

$$L(f(x), y) \tag{2.2}$$

quantifies the error of the prediction $f(x)$ when the true output is y. The choice for loss function depends on the task, Hinge Loss[1] is common for classification with Support Vector Machines (SVM), while the square loss (or L2 loss) is popular within regression tasks, in Deep Learning classification algorithms, the cross-entropy loss is the most commonly used one (i.e., for Softmax output layer when dealing with multi-class problems).

Given some function f, a loss function L and the true distribution $p(x, y)$ over inputs x and y we can define the expected or (true) Risk as follows:

$$R_{true}(f) \triangleq \mathbb{E}[L(f(x), y)] = \int \int p(x, y) L(f(x), y) dx dy. \tag{2.3}$$

The risk measures how much, on average, it costs to use f as our prediction algorithm. The idea is to make R_{true} small, however the main problem here is that we don't know μ we defined earlier (i.e., the true distribution of our data points). We can approximate this by the empirical error; Given some function f, a loss function L and a training set S that is made by n data points, we can now define the empirical risk on that training set as:

$$R_{emp}(f) \triangleq \frac{1}{n} \sum_{i}^{n} L(f(x_i), y_i). \tag{2.4}$$

In the above definition, it is typical to include a regularization parameter G, such as:

$$R_{emp}(f) = \underbrace{\frac{1}{n} \sum_{i}^{n} L(f(x_i), y_i)}_{\text{Risk (training error)}} + \underbrace{\lambda G(f)}_{\text{Regularizer}} . \tag{2.5}$$

This parameter is included in order to impose a complexity penalty on the loss function and prevent overfitting (i.e., it tries to implement Occam's Razor, the simpler the model the better), where $G(f)$ measures the complexity of the prediction function f and λ controls the strength of the complexity penalty.

The goal in this case, is, to *minimize* this quantity, that is, we need to find that $f^* \in \mathcal{H}$ which minimizes the empirical risk:

$$f^* = \arg \min_{f \in \mathcal{H}} R_{emp}(f). \tag{2.6}$$

Restricting the space of functions $\mathcal{H}$ to those parametrized by θ, we can re-write Equation 2.6 as follows:

$$\theta^* = \arg \min_{\theta \in R^d} \left\{ \frac{1}{n} \sum_{i}^{n} L(f(x_i; \theta), y_i) + \lambda G(\theta) \right\} \tag{2.7}$$

[1] $L(f(x; \theta), y) = \max(0, 1 - y \cdot (\theta^\top x))$, where $\theta \in R^d$, includes all learnable parameters.

Where θ, accounts for all learnable parameters. Therefore, solving the aforementioned regularized convex optimization problem and finding θ^*, leads to the "learnt model".

We note that under certain loss functions, for example, the Negative Log-Likelihood loss (NLL), which is widely used for training Deep Neural Networks, the Empirical Risk Minimization is basically reduced to Maximum Likelihood Estimation (MLE).

More concretely, suppose that our loss function acts like a posterior density $\mathcal{Q}$, with parameterization θ, given some data points (x, y), that is $\mathcal{Q}(\theta|x, y)$. Let's re-define the Loss in Equation 2.2 as follows:

$$L(f(x;\theta), y) \triangleq -log(\mathcal{Q}(x, y|\theta)) \tag{2.8}$$

which we will refer to it as negative log-likelihood loss. We can show that minimizing the (non) regularized empirical risk (Equation 2.4) under this loss function (Equation 2.8) is equivalent to estimating the maximum likelihood[2]:

$$\arg\min_{\theta \in R^d} \sum_{i}^{n} (-log(\mathcal{Q}_\theta(x_i, y_i|\theta))) = \arg\max_{\theta \in R^d} \sum_{i}^{n} log(\mathcal{Q}(x_i, y_i|\theta)). \tag{2.9}$$

In the above formulation a scaling factor $\frac{1}{n}$ could be added to make it clearer as this does not affect the argmin/argmax of the log probability.

Starting from our initial assumption of the posterior and expressing it by Bayes:

$$\mathcal{Q}(\theta|x, y) = \frac{\mathcal{Q}(x, y|\theta)\mathcal{Q}(\theta)}{\mathcal{Q}(x, y)} \tag{2.10}$$

then by taking the logarithm of this expression:

$$log\mathcal{Q}(\theta|x, y) = log\mathcal{Q}(x, y|\theta) + log\mathcal{Q}(\theta) - log\mathcal{Q}(x, y) \tag{2.11}$$

we can now omit the last factor since it is a constant (does not depend on θ):

$$log\mathcal{Q}(\theta|x, y) = log\mathcal{Q}(x, y|\theta) + log\mathcal{Q}(\theta). \tag{2.12}$$

Thus, if we would like to maximize this posterior such that we obtain the optimal model parameters θ^* over all training samples (x_i, y_i), we can write:

$$\theta^* = \arg\max_{\theta \in R^d} \Big\{ \underbrace{\sum_{i}^{n} log(\mathcal{Q}(x_i, y_i|\theta))}_{\text{Likelihood}} + \underbrace{log(\mathcal{Q}(\theta)}_{\text{Prior}} \Big\}. \tag{2.13}$$

In case the last term, the prior, is uninformative (i.e., $\mathcal{Q}(\theta) = 1$) we return to the original formulation of MLE (Equation 2.9). Otherwise, this is called Maximum-a-Posteriori Estimation (MAP), as the role of the prior is basically

[2]It is the probability of the data (D), given the parameters (θ). In the logarithmic domain we define it as follows: $log\mathcal{L}(\theta) = logP(D|\theta) = logP(d_1, ..., d_n|\theta) = log\prod_i^n P(d_i|\theta) = \sum_i^n logP(d_i|\theta)$.

to encode some prior knowledge about which models are more or less likely. Under the ERM framework, this prior can be interpreted as the regularization term, while the likelihood can be interpreted as the empirical risk. Finally, we note that depending on the distribution that the error/risk is modeled around, we obtain different estimators; minimizing the log-likelihood under Gaussian distribution results in a Least Squares loss, while a Bernoulli distribution would give the Binary Cross-entropy loss.

Optimization

These kinds of optimization problems are usually solved by Gradient Descent (GD) algorithms that use the slope of the surface of the loss function to find some optimal region, where the error is minimized. In the case of Deep Neural Networks (DNN), strictly speaking, we usually end up with highly non-convex loss surfaces [15] (i.e., we can get multiple local minima [12]). Thus we use Stochastic Gradient Descent (SGD) class solvers, which in their simplest form, they just shuffle the samples and update the gradient after each example. Practically, a variation of this method is widely used where the datapoints are basically packed and aggregated into mini-batches to update the gradient at each iteration (mini-batch SGD) [61]. Empirically speaking, this method considerably speeds up the training procedure (over GD), and usually converges to reasonably good solutions.

Briefly, SGD is an iterative algorithm for solving the (regularized) convex optimization problem in Equation 2.7. It begins from some initial model parameters θ_0, and, at every step t, it updates the model parameters as follows:

$$\theta_{t+1} = \theta_t - \eta(\lambda \nabla G(\theta_t) + \nabla L(f(x_t; \theta_t), y_t)), \tag{2.14}$$

where η is the learning rate, $\nabla G(\theta_t)$ is the gradient of the regularizer, and $\nabla L(f(x_t; \theta_t), y_t)$ is the gradient of the loss function, evaluated on a single example (x_t, y_t). In the case of DNN's, this gradient can be calculated using automatic differentiation (such as the Backpropagation algorithm, popularized by [85]) with respect to each layer's learnable parameter set θ^{layer}.

2.2.1 Neural Networks

Artificial Neural Networks (ANN), make up a class of algorithms that can be used both for Supervised learning tasks (e.g., classification [55], regression [100]) and Unsupervised learning tasks (e.g., clustering [54] and even dimensionality reduction [64]). Neural Networks, deep or shallow, are typically constructed and understood around the empirical risk minimization (ERM) principle [105], which we briefly introduced above. Their name is inspired by the biological neural networks due to similarities in their structure; however, they are not directly comparable.

A motivating fact to study and apply Neural Networks stems from the Universal Approximation Theorem (UAT) [17, 44]. According to it, informally speaking, a feed-forward[3] neural network can approximate any function as long as it has

[3] An acyclic network where information flows from input all the way to output.

at least one hidden layer[4]. This is a rather bold statement as it implies that any problem that can be reduced to function approximation could be potentially solved by a neural network with just a single hidden layer.

Goodfellow et al. [35] add to this statement and note that: *the [hidden] layer may be infeasibly large and may fail to learn and generalize correctly.* In other words, some functions might be too computationally "hard" to approximate using neural networks as the network's hidden layer has to be rather wide. This is formally shown in recent works that suggest that the width (number of nodes) of such networks has to be even *exponentially large* in the dimension (and thus potentially impossible to calculate) to be able to approximate some functions with just one hidden layer [28, 62, 37].

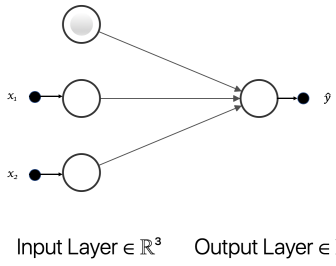
Structure of Artificial Neural Networks

Artificial Neural Networks can be understood as a series of consecutive differentiable functions composition. Hence, naturally, ANNs can be represented by layered graphs (network topologies). The most popular ones are arguably the feed-forward neural networks (FNN). As their name implies, they are made up of directed and acyclic network topologies, where the data flows from input all the way to output. Perceptron, proposed by Rosenblatt [83], is the simplest form of a FNN with two layers only (input layer & output layer), Figure 2.1a . Perceptron implements a binary linear classifier characterized by Equation 2.15:

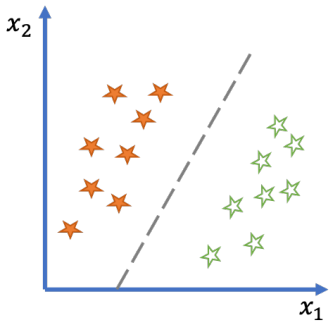
$$\phi(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{w}^\top \mathbf{x} + b > 0, \\ 0 \text{ otherwise} \end{cases} \tag{2.15}$$

where $\phi(\cdot)$ called the *activation function*, $\mathbf{x}$ accounts for the input vector, $\mathbf{w}$ and b accounts for the set of learnable parameters, *weights* and *bias* respectively. Intuitively, for some d-dimensional classification problem, we would like to find a decision hyperplane (a line in the 2-dimensional setting, Figure 2.1b) that best separates one class from another. This can be realised after "learning" the correct parameters (bias and weights). Weights are responsible for rotating the separation hyperplane, while bias is responsible for shifting it w.r.t. origin, until *all* examples are classified correctly.

[4]In fact, UAT is more restrictive as it requires the layer to be non-linearly transformed (realized via the introduction of non-linear activation functions). In addition, it states that an "adequate" number of nodes is needed in order to approximate a function up to a desired precision.



(a) Perceptron architecture. Black nodes correspond to inputs and outputs, the filled input with grey color corresponds to the bias parameter, while, white nodes are called neurons.



(b) A linearly separable problem.

Figure 2.1: The Perceptron and types of problems it can deal with.

The main problem with Perceptrons (and heavily criticised in [70]), is their inability to find non-linear decision boundaries; If the problem is not linearly separable, they will never converge. *Multi layered networks* can overcome this issue. In this case, we have feed-forward neural networks with multiple layers between the input layer and output layer (i.e., *hidden layers*) with non-linear activation functions. Intuitively, it can be thought as multiple Perceptrons organised in layers with non linear activation functions, Figure 2.2.

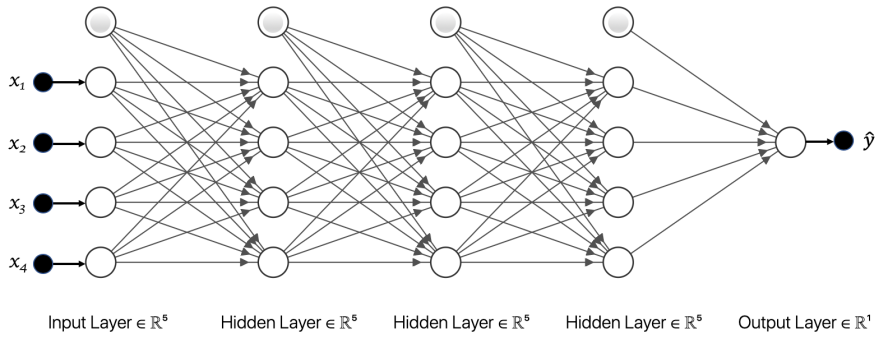


Figure 2.2: A Multi layered network. Black nodes correspond to inputs and outputs. Filled inputs with grey color correspond to the bias parameters, while the empty white nodes are called neurons. The edges between the nodes make up the set of learnable parameters of such networks.

These non-linear activation functions are of vital importance for modeling non-linear relationships within the data. For example, we can consider a multi-layered network with linear activation functions. The composition of such a network is basically a linear combination of all of its layers, and thus it can be reduced to a network without hidden layers (only input & output layers) such

as a Perceptron. For the universal approximation theorem to be valid, and so we are able to harness the expressive power of neural networks, it is required that the ANN is basically a composition of non-linear activation functions and not only arbitrarily deep (i.e., more hidden layers). In the early works of neural networks, the community mainly used the hyperbolic tangent and the sigmoid as activation functions to introduce the much-needed non-linearities. More recently, a class of *rectifier* activation functions became increasingly popular (such as ReLU [74], PReLu [40], ELU [16], etc.), especially with deeper networks, as they practically allow a faster convergence to the objective, and help alleviate issues involving gradient computations[5].

Common Types of Layers

In this section we briefly introduce the building blocks we used for this thesis.

A typical way to build simple, layered neural networks is by adding *Fully Connected* (FC) layers, as in Figure 2.2. In FC topologies, each node takes as an input the linear combination of all outputs from the previous layer. This is a generic type of layers that can be used either standalone or, they can be combined with other types of layers for more sophisticated ML tasks (e.g., Convolutional Layers for involved visual recognition tasks).

Convolutional Layers is another very widely used type of layers, primarily applied for visual related [58, 55] tasks; however, their capabilities extend well beyond the visual domain, for example, to natural language processing and speech recognition [52, 115, 116]. In this case, input data is convolved with a kernel (much like the way it is done in image processing), and since this is a spatial operation, the input data is usually structured in a grid-like format such as images, time-series, etc. These kernels are usually referred to as "filters", and they make up the set of learnable parameters of the layer. Contrasting to the FC layers, where each node is "connected" to every output from the previous layer, convolutional networks implement what is called *sparse connectivity*.

This is realised as the convolved filters are significantly smaller than the input volume, which basically results in a reduced number of connections between the input and the output layer (the output layer in convolutional networks is also referred to as a *feature map*). Additionally, since each element of the filters is being convolved with the input volume more than once (in order to cover the entire input volume), the parameters (weights) of the convolutional layer are shared across different regions of the input volume. Practically, parameter sharing and sparse connectivity result in a more efficient network with a decreased number of total parameters. Therefore the amount of computations and memory needed is significantly smaller compared to FC layers. Another characteristic property of the Convolutional layers is that they ensure equivariance to translation (i.e., if we translate the input volume, the resulting feature map is translated in the same way) which arises from the way convolution is realized (i.e., by sliding the filter over the input volume).

[5] Refers to the vanishing gradient problem [6, 43].

It is also worth noting that these filters tend to learn automatically hierarchical feature representations from the inputs. In the context of image inputs, filters from early convolutional layers tend to learn generic primitives such as lines, edges & color gradients, while, as more convolutional layers are added, their filters tend to learn more specific features depending on the input data (e.g., ears, eyes, legs) [60]. Typically, a (max) pooling operation is performed after the convolution to downsample the output. This operation, adds the property of translation invariance to convolutional networks, in the sense that a slight translation of the input does not (or only marginally) change the output [117].

Pooling has proved to be a very desirable effect in most object recognition tasks; however, one major drawback of this, is that it disregards the relative order of the objects within the input volume. Roughly speaking, given an image volume, that has random, unordered features in it, such as human parts (like a leg, a nose, an eye), it is very likely that a neural network based on convolutional layers might be "fooled" that this is an image of a human. Capsule and Spatial transformer networks proposed in [86, 41, 46] provide a framework to alleviate this issue.

Training Neural Networks

The main idea, following the ERM paradigm, is to find a way to approximate the empirical distribution (i.e., the function that underlines the *given* training data) by the model's distribution (i.e., the one defined by the learnable parameters of the neural network). One way of addressing this problem is by minimizing the distance between these two distributions. Minimizing the Kullback–Leibler (KL) divergence does exactly this in the form of the popular Cross-entropy loss function (we note that this is equivalent to maximizing the log-likelihood discussed earlier).

Cross-entropy loss measures this distance (or error) in terms of how the network's predicted output differs from the desired ground truth. Since the output is a function composition of the entire network parameters, minimizing this error implies computing the partial derivatives of this loss function with respect to every layer's learnable parameters. The error-back-propagation algorithm, introduced by Werbos [108] provides an efficient way of computing this gradient in linear time using the chain rule. This is usually paired with gradient descent class optimizers (e.g., Adam [53], Adagrad [20], Adadelta [112]) to follow the steepest direction of the gradient where the error is supposed to be small.

Depending on the task we would like to solve, the output layer has to be constructed with a sufficient number of nodes and a suitable activation function. For example, in the binary classification setting, one node is sufficient with a sigmoid activation function. Whereas, in multi-class settings, n output nodes are required for an n-class problem ($n > 2$). In this setting, a generalization to sigmoid, the Softmax activation function is used, which help us interpret the output as a probability distribution over the given classes.

Federated Learning

Over the last years, there has been an increasing interest in machine learning solutions that can be trained locally, for example, on handheld devices, such as mobile phones, wearables, etc. Practically speaking, this is because a lot of the training workload can be outsourced to millions of capable devices around the world. Training models locally, also ensures (to some extent) fewer privacy violations for the participating entities since their data never leave their devices. On the other hand, training models centrally (in some central computer) implies certain disadvantages. Apart from the obvious risk of raw data sharing, it requires the use of powerful (and rather expensive) computers. In addition, one could even consider costs that involve managing all clients' data (e.g., uploading and storing them to some central computer).

McMahan et al. in [65] proposed the concept of Federated Learning to address this issue (see Algorithm 1). In this case, the dataset is distributed to K clients, where each one has to train on its local data partition and create its local model (e.g., by using SGD). Once the local training round has been finished, all the clients' models are then gathered by the central server to aggregate them (i.e., the server is averaging out all the models' parameters). This way, the server comes up with a new central model, without performing the actual training part (i.e., the server never computes SGD steps over all data points).

Experiments show that this relatively naive method works surprisingly well, both for i.i.d. and non-i.i.d. sample distributions (with the latter being a more realistic scenario for FL applications). The utility of Federated Learning has been explored in image classification [65], in mobile keyboards for next word prediction (auto-complete) [38], or even for emoji characters prediction with high success, such as in [82].

Algorithm 1 Federated Averaging, [65]. The K clients are indexed by k; B is the local minibatch size, C is the fraction of participating clients, E is the number of local epochs, $\mathcal{P}_k$ is the local data partition of client k and η is the learning rate.

Server Executes:
initialize w_0
for each round t =1,2,... **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): //Runs on Client k
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Although Federated Learning *can* provide some privacy, in the sense that the user's data never leave from their devices, there are still some significant risks concerning the individual's privacy. In machine learning and particularly in (deep) neural networks, it is quite typical that the models can memorize the actual data examples internally [2, 113]. This happens mainly due to their rather large amount of free parameters (commonly referred to as overfitting); Federated Learning is no exception to this.

For example, in the auto-complete scenario, it might be rather convenient to get auto-completion for the exact credit card details (e.g., name, surname, or even the actual credit card number) when it is needed to perform some (online) transaction. However, this means that some parts of the local model parameters have memorized these data points. In addition to this, they have been shared with the central server in the form of updated gradients. This is a clear violation of privacy as it can potentially leak even more personal data details (and is also prone to model inversion attacks).

On the bright side, there is a well established mathematical and statistical set of techniques that can help alleviate these kinds of problems in a very formal and well-defined way. Differential Privacy (and related techniques that work under this notion) is able to provide rigorous mathematical bounds so that the aggregated model under these bounded constraints will not be able to memorize and leak the private data of the participating entities.

2.3 Differential Privacy

Differential Privacy (DP) by Dwork et al. [23, 21, 24] comes to address these kinds of problems. DP has received increasing attention recently as it provides a rigorous and well defined statistical framework for privacy; It can limit the release of sensitive information derived from private data using some auxiliary information as in the IMDB case study. DP algorithms rely heavily on noise injection so that the adversary receives something noisy, in such a way that a potential victim cannot be attributed to having some property $\mathcal{A}$. For example, suppose that a potential adversary has access to a database or dataset (we will use these two terms interchangeably from now on) of medical records. If the queries made to the database by the adversary are bounded by a DP guarantee, the adversary should be able to find what is the average age of cancer patients, however, he should not be able to attribute cancer to some person within the database (even if the adversary possesses auxiliary information about the patients).

Differential Privacy can be understood as a probabilistic concept for privacy protection. It can be realized by introducing randomness to sensitive data functions (e.g., Database queries or ML models). To put this into perspective, we can think for a moment of Randomized Response [107]. This is a technique that developed before DP and used widely in the past, especially in social sciences, to construct *plausible deniability* for the subjects who took part in controversial and/or embarrassing experiments (crowd-sourced statistics).

A simple algorithm that demonstrates whether a participant of some crowdsourced statistic has answered "yes" or "no" to a controversial property P via the randomized response can be seen in Algorithm 2.

Algorithm 2 Algorithm - Randomised Response

```
flip a coin
if (coin == tails):
    answer Yes
else:
     answer Truthfully
```

To put this algorithm into perspective, suppose that we would like to perform a survey in a penitentiary institution to find out whether the inmates are trafficking drugs within the facilities. Due to the sensitivity of the question, it is highly likely that the inmates will not answer honestly. To protect their privacy, we could instead ask them to flip a coin in private. Then, if the coin is tails, answer "yes", if the coin is heads, answer the question "Are you drug trafficking" truthfully. From this randomized process, we can see that even if the subject has answered "yes" for having the controversial property P, we cannot really attribute it with the property since most would be answering "yes" anyway because of getting tails on the coin flip.

To illustrate this example, suppose that we survey 100 inmates using Algorithm 2, and, as a result of this, it turned out that 55% of the inmates answered "yes" for drug trafficking. Since the coin is supposed to be fair (i.e., $p_{tails} = p_{heads} = 0.5$), we would expect that half of the inmates, that is 50, would have got tails and thus answered "yes" no matter if they committed the crime or not. The excess of 5 "yes" answers indicates that 5 out of 50 (10%), of those who were expected to receive heads (i.e., truthful answers), have actually committed the crime. In addition, we would expect that there are also 5 drug traffickers amongst the 50 who received tails and answered yes anyway. So finally, we can say that 10 out of 100 (10%) of the inmates surveyed are involved in illegal activities. Strictly speaking, this algorithm protects the privacy of those who answered "yes" as we can still tell that those who answered "no" they definitely don't possess the embarrassing attribute and this fact by itself it constitutes a privacy breach to some extent. More involved versions of this mechanism can be found in [24, 107] that address this issue.

This analysis, demonstrates that although we have managed to inject a relatively high amount of randomness in the recorded answers of *each* subject, we can still achieve 1) High degree of anonymity for the participants (i.e., we cannot attribute a higher probability to any participant of those who answered "yes" for having the controversial property P) and 2) We are still able to extract meaningful and accurate statistics over the *aggregate* population under examination. It is worth mentioning, that the added privacy from this type of mechanisms comes at the cost of accuracy; Since these mechanisms take the true statistic and average it with a 50-50 coin flip, there is still a chance that the participating individuals might flip the coin in a very unlikely way such that the results are skewed. That basically means that a rather large amount of participants

have to be involved in these kinds of studies to be able to extract the accurate distribution.

Definition of Differential Privacy

Definition 1 (Differential Privacy)**.** *A randomized mechanism $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ with a domain $\mathcal{D}$ (e.g., all possible training datasets) and range $\mathcal{R}$ (e.g., all possible trained models) satisfies (ϵ, δ)-differential privacy if for any two adjacent datasets d, d' $\in \mathcal{D}$ and for any subset of outputs $\mathcal{S} \subseteq \mathcal{R}$ it holds that: $Pr[\mathcal{M}(d) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathcal{M}(d') \in \mathcal{S}] + \delta$.*

Where ϵ is the privacy budget (the smaller the ϵ, the more private the mechanism is), δ is the probability of the privacy mechanism failure. If δ is 0, then we say that $\mathcal{M}$ is $\epsilon-$differentially private. By adjacent datasets ($d \simeq d'$), we can consider, for example, that two training datasets (made of image-label pairs) are adjacent if they differ in a single entry, that is, if one image-label pair is present in one set and absent in the other.

Privacy Loss

Let $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ be a randomised mechanism, with density function $Pr[\mathcal{M}(d) = r]$, we can then define the Privacy Loss function as follows:

$$\mathcal{L}_{\mathcal{M},d,d'}(r) = ln\left(\frac{Pr[\mathcal{M}(d) = r]}{Pr[\mathcal{M}(d') = r]}\right) \tag{2.16}$$

with neighboring datasets $d, d' \in \mathcal{D}$ and an outcome $r \in \mathcal{R}$. We can treat Privacy Loss as a random variable (rv) which depends on the random noise added to the algorithm. The privacy loss rv can be defined as:

$$L_{\mathcal{M},d,d'} \triangleq \mathcal{L}_{\mathcal{M},d,d'}(r). \tag{2.17}$$

We can then say that a mechanism $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ is (ϵ, δ)-differential private if for any $d \simeq d'$ we have that $Pr[L_{\mathcal{M},d,d'} \geqslant \epsilon] \leqslant \delta$.

Properties of Differential Privacy

Differential Privacy has a set of properties making it appealing to work with:

- **Composition**. Each time we query the database or the dataset to gather some new insight, or, a statistic of the population, there is a possibility that we might learn something more about the participating subjects. Therefore, after each time we ask a new query at some step T, we need a way to quantify this privacy degradation.

  Sequential composition, [24]: In this case, if we have k mechanisms, M, which are each (ϵ, δ)-differentially private, we would like to be able to use the outputs from the first into the second, and so on, without completely sacrificing privacy. That is, for $i \in \{1, 2, ..., k\}$, let $M_i(d)$ be an (ϵ_i, δ_i)-differentially private mechanism executed on database d. Then, the function composition $\mathcal{F}$ of these mechanisms, that is $\mathcal{F} = (M_1 \circ M_2 \circ ... \circ M_k)$, is $(\sum_i \epsilon_i, \sum_i \delta_i)-$differentially private.

Parallel composition, [24]: As we can see above, the serial composition assumes that the outputs are correlated which results in a more pessimistic total privacy budget value ϵ and a higher probability of failure δ; that is, the privacy can be severely degraded after many applications of different mechanisms M on the dataset. In parallel composition, we consider the situation where we have a single database, d, partitioned into k disjoint subsets, d_i. Then, if we have $M_1, M_2, ..., M_k$ mechanisms computed on these disjoint subsets (i.e., $M_i(d_i)$), with privacy guarantees $\epsilon_1, \epsilon_2, ..., \epsilon_k$ and $\delta_1, \delta_2, ..., \delta_k$ respectively, then any function composition $\mathcal{F}$, of these mechanisms, $\mathcal{F} = (M_1 \circ M_2 \circ ... \circ M_k)$ is $(\max_i \epsilon_i, \max_i \delta_i)-$differentially private.

Advanced composition. In addition to the basic composition versions introduced above, more involved versions have been suggested, which significantly improve the privacy cost. For example, in [26], authors improve the privacy cost over sequential composition and show that after T steps, $(\epsilon\sqrt{Tlog(1/\delta)}, T\delta) - DP$ can be achieved. This can be further improved using sampling methods as in [50].

Accountants. This is a different approach to keep track of privacy spending. It has been proposed in [68], and more recently, Abadi et al. [1] used it for training neural networks. Authors treat the privacy loss as a random variable, and then, they use a moment-generating function to calculate higher moments of this rv, which then they bound to show that they can provide a $(q\epsilon\sqrt{T}, \delta)-$DP guarantee, where q is a sampling rate.

Example: Suppose that we have a mechanism where each step is $(1.3, 10^{-4})$-DP and we would like to compose it for $T = 1000$ steps. Under sequential composition this would give $(1300, 0.1)$-DP, under the advanced composition we would get $(125, 0.1)$-DP, while, using Abadi's accountant we would get $(4.1, 10^{-4})$-DP for a sampling rate of $q = 0.1$, which is a significantly better guarantee.

- **Independence from Auxiliary Information**. Differential Privacy guarantees that if an adversary is holding auxiliary information, as in the Netflix-IMDB example, it doesn't increase their chances of a successful attack (including all those attempted with all past, present, and future datasets) beyond the provided guarantee.

- **Immunity to Post Processing**. Any function (independent of data) of the output of a differentially private algorithm is also differentially private. That is, the adversaries cannot increase the privacy loss of the algorithm by using this output in any way (privacy guarantees hold as long as the adversaries don't have any knowledge of the *original* private data).

  More formally: Let $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ be an (ϵ, δ)-differentially private mechanism, and let $f : \mathcal{R} \mapsto \mathcal{Y}$ be any function of $\mathcal{R}$, that takes as input only the output of the mechanism $\mathcal{M}$ *and* is independent of the data, then $f(\mathcal{M}) : \mathcal{D} \mapsto \mathcal{Y}$ also preserves (ϵ, δ)-differential privacy [84].

2.3.1 Privacy Mechanisms

One way of designing differentially private mechanisms, is, by adding *calibrated* noise drawn from certain probability distributions to "sensitive" functions. By calibration, we refer to the "adjustment" of the standard deviation of the noise according to the *sensitivity* of the function f. Where, f is a function of the data (e.g., a classification algorithm or a database query). Intuitively, sensitivity measures how much would the function's f output can change after we perturb any single input over its domain. Roughly speaking, if the data is very sensitive, we would expect the adjustment done to the standard deviation to be quite significant (i.e., it would end up adding more noise to compensate for it).

The Gaussian Mechanism

The so-called Gaussian Mechanism (GM), which depends on the ℓ_2-sensitivity function, is one popular way to achieve Differential Privacy. The Gaussian Mechanism approximates a deterministic, real-valued function $f : \mathcal{D} \mapsto \mathcal{R}$ with a differentially private mechanism by adding noise that is calibrated to f's sensitivity. The sensitivity, S_f, is defined as the maximum of the absolute distance between two adjacent inputs d and d' i.e., $S_f = ||f(d) - f(d')||_2$. Therefore, we can now define the GM as follows:

$$\mathcal{M}_G(d) \triangleq f(d) + \mathcal{N}(0, S_f^2 \cdot \sigma^2) \tag{2.18}$$

where, $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ is the Gaussian Distribution[6] with zero mean and standard deviation $S_f \cdot \sigma$ In this case, it is shown in [24] [Theorem 3.22], that a single application of this mechanism for any $d \simeq d'$, with sensitivity S_f, it is said to be (ϵ, δ)-private if $\delta \geqslant \frac{4}{5}\exp(-(\sigma\epsilon)^2/2)$ and $\epsilon < 1$.

The Laplace Mechanism

The Laplace Mechanism (LM) works similarly to the GM, however, the noise is drawn from the Laplace distribution. LM depends on the ℓ_1-sensitivity function, that is, $S_f = ||f(d) - f(d')||_1$, and is defined as:

$$\mathcal{M}_L(d, \epsilon) \triangleq f(d) + \text{Lap}\left(0, \frac{S_f}{\epsilon}\right) \tag{2.19}$$

where $\text{Lap}\big(0, \frac{S_f}{\epsilon}\big)$ is the Laplace Distribution[7], centered around zero. We notice that for a weak privacy guarantee (i.e., large value for ϵ) this mechanism adds less noise, while, more noise is added when the privacy guarantee is strong (i.e., small value for ϵ). Laplace Mechanism provides a stronger privacy guarantee compared to GM, as it preserves $(\epsilon, 0)$-differential privacy (proof in theorem 3.6, in [24]).

[6] The probability density of the 1-d Gaussian Distribution with mean μ and standard deviation σ, is given by $f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

[7] The probability density of the Laplace Distribution with mean μ and scale b, is given by $f(x \mid \mu, b) = \frac{1}{2b} e^{\left(-\frac{|x-\mu|}{b}\right)}$. The variance for this distribution is given by $\sigma^2 = 2b^2$.

2.3.2 The Moments Accountant

As we noted already, the parallel composition theorem generally provides a pessimistic guarantee on privacy loss after each query is executed. In fact, it is very loose as it only assumes a worst-case scenario. Thus, it is possible to come up with methods that can provide stronger privacy guarantees. Abadi et al. [1] propose the *Moments Accountant*, to address this issue. This method is inspired by an earlier work, [68], which provides a framework for budgeting privacy, creating and enforcing privacy policies in the context of differential privacy. It helps to create dp-queries and basically to keep track of the repeated applications of additive noise mechanisms.

Roughly speaking, it allows the development of methods such as [1], so that one can choose in advance some privacy budget ϵ that allows the execution of queries until this budget is depleted. In other words, Abadi's accountant allows one to pick some desired ϵ value for the privacy budget, and then, it can find the minimum δ that corresponds to it (and vise-versa).

The Moments accountant can be understood under the notion of *Rényi Differential Privacy* (see Mironov, [71]) which basically generalizes the notion of (ϵ, δ)-differential privacy we presented earlier (Definition 1). Before going into the details of the Moments Accountant, we briefly introduce Rényi Differential Privacy (RDP) and additional useful material from [71].

Definition 2 (Rényi Divergence)**.** *For two probability distributions P and Q defined over $\mathcal{R}$, the Rényi divergence of order $\alpha > 1$ is defined as follows:*

$$D_\alpha(P||Q) \triangleq \frac{1}{\alpha - 1} \ln E_{x\sim Q}\left(\frac{P(x)}{Q(x)}\right)^\alpha \tag{2.20}$$

with $P(x)$ denotes the density of P at x. We note that while the parameter α is approaching one, it corresponds to KL divergence (by taking the expectation with respect to P in Equation 2.20) which is a measure of "closeness" of two distributions. Similarly, Rényi Divergence, measures the closeness of the two distributions by taking into consideration α, which for different values, it provides different kind of information about the "similarity" or "closeness" of the distributions.

Definition 3 ($(\alpha, \epsilon)-$Rényi Differential Privacy [71])**.** *A randomized mechanism $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ is said to have $\epsilon-$Rényi differential privacy of order α, or (α, ϵ)-RDP for short, if for any adjacent $d, d' \in \mathcal{D}$ it holds that $D_\alpha = (\mathcal{M}(d)||\mathcal{M}(d')) \leq \epsilon$.*

Lemma 1 (Adaptive Sequential Composition in Rényi-DP [71], Proposition 1)**.** *Let the mechanisms $\mathcal{M}_1 : \mathcal{D} \mapsto \mathcal{R}_1$ be (α, ϵ_1)-RDP and $\mathcal{M}_2 : \mathcal{M}_1 \times \mathcal{D} \mapsto \mathcal{R}_2$ be (α, ϵ_2)-RDP, then, their composition satisfies $(\alpha, \epsilon_1 + \epsilon_2)$-RDP.*

Lemma 2 (From RDP to (ϵ, δ)-DP [71], Proposition 3)**.** *If $\mathcal{M}$ is an (α, ϵ)-RDP mechanism, it also satisfies $(\epsilon + \frac{\ln(1/\delta)}{\alpha - 1}, \delta)$-differential privacy for any $0 < \delta < 1$.*

Moments Accountant

The idea behind the moments accountant is based around bounding the moments of the log of the moment generating function (MGF) in order to find

some optimal ϵ or δ value given some δ or ϵ respectively. Roughly speaking, this can be realized in three steps:

1. Compute λ moments of the log MGF of the additive noise mechanisms.
2. Compose the mechanisms.
3. Find the optimal ϵ, or δ.

In order to compute the moments, the authors choose to pose the problem in terms of treating the privacy loss as a random variable, L (see Equation 2.17), and compute the log MGF of that variable. The idea is to keep tracking a bound of the privacy loss rv instead of a bound on the original privacy budget. Generally speaking, interesting statistical properties (such as mean, variance, etc.) for some rv, X, can be described by finding its moments, and, the probability distribution of such rv, X, can be understood by its MGF ($\text{MGF}_X(\lambda) = E[\exp(\lambda X)]$).

More concretely, the log MGF of the privacy loss rv can be defined as: $\Lambda_{\mathcal{M},d,d'}(\lambda) \triangleq \ln E[\exp(\lambda L_{\mathcal{M},d,d'})]$ which we can re-write as:

$$\begin{aligned}\Lambda_{\mathcal{M},d,d'}(\lambda) &= \ln E_{r\sim\mathcal{M}(d)}\left(\frac{\mathcal{M}(d)(r)}{\mathcal{M}(d')(r)}\right)^{\lambda} \\ &= \ln E_{r\sim\mathcal{M}(d')}\left(\frac{\mathcal{M}(d)(r)}{\mathcal{M}(d')(r)}\right)^{\lambda+1} \\ &\quad \text{(second equality follows by change of measure,} \\ &\quad \text{see also Appendix of [1]).}\end{aligned} \tag{2.21}$$

We can now see that from Definition 3 and Equation 2.20, 2.21, that (α, ϵ)-RDP can be written in terms of the log MGF of the privacy loss rv as follows:

$$D_\alpha = \frac{1}{\alpha - 1}\Lambda_{\mathcal{M},d,d'}(\alpha - 1) \leq \epsilon. \tag{2.22}$$

So, on above Equation, as $\lim_{a\to 1}$, moments accountant would basically get a bound on the expectation of the privacy loss rv, while under certain conditions (see [71]), if $\lim_{a\to\infty}$ then we get pure differential privacy (i.e., $(\epsilon, 0)-$dp). More formally, in order to bound the moments of this rv, we define the Moments Accountant $\mathcal{C}$ for a mechanism $\mathcal{M}$ at some λ as follows:

$$\mathcal{C}_{\mathcal{M}}(\lambda) \triangleq \sup_{d,d'} \Lambda_{\mathcal{M},d,d'}(\lambda) \tag{2.23}$$

assuming that d and d' differ at most by one entry. For each mechanism, we basically evaluate $\mathcal{C}_{\mathcal{M}_i}$ at a list of predefined λ values and then, we use Lemma 1 in order to compose the mechanisms and bound the moments of the (composite) mechanism overall (for each λ). Finally, by Lemma 2 we can translate it into (ϵ, δ)-dp and find the lowest δ for some given ϵ (and vise versa) by using:

$$\epsilon(\delta) = \min_{\lambda} \frac{\ln(1/\delta) + \mathcal{C}_{\mathcal{M}}(\lambda)}{\lambda} \tag{2.24}$$

$$\delta(\epsilon) = \min_{\lambda} \exp(\mathcal{C}_{\mathcal{M}}(\lambda) - \lambda\epsilon). \tag{2.25}$$

Thus, solving the optimization problems posed above (Equations 2.24, 2.25) (either in some closed form, or, by searching over the space of λ's) leads to the desired ϵ or δ values. For more details and derivations, see [1, 71].

2.4 Machine Learning & Differential Privacy

We can apply differential privacy mechanisms within empirical risk minimization (ERM) framework in different ways. One approach is to *perturb the objective*, that is, the Equation 2.5 becomes:

$$R_{emp}(f) = \frac{1}{n}\sum L(f(x_i), y_i) + \lambda G(f) + \textbf{noise}. \tag{2.26}$$

where the noise is always calibrated according to some data sensitivity S_f. Therefore, in this case we basically perturb the optimisation surface, and then optimize, such as in [51, 14]. It is worth noting that this approach can be restrictive as it assumes twice-differentiable loss functions and strong convexity.

Gradient perturbation is another way to add privacy to ERM framework. One such example is by Song et al. [92] and Bassily et al. [5], where they show how to keep the model differentially private by injecting noise (calibrated to data sensitivity) directly on the gradients:

$$\theta_{t+1} = \theta_t - \eta(\lambda\nabla G(\theta_t) + \nabla L(f(x_t;\theta_t), y_t) + \textbf{noise}). \tag{2.27}$$

In this case, they achieve differentially private updates to the model at each step. According to authors [92], with normal SGD (i.e., model gets updated after each sample), this becomes infeasible as the added variance from noise injections after each step increases to the point that the algorithm does not converge. Instead, they suggest batched computations of SGD (i.e., more samples per update round) that seem to mitigate this problem and perform well accuracy-wise.

Another convenient way to achieve differential privacy within ERM is to add noise at the *output of the optimization process*. That is, we first minimize Equation 2.7 and then add calibrated noise to it [110, 48]:

$$\textbf{noise} + \arg\min_{\theta\in R^d}\left\{\frac{1}{n}\sum L(f(x_i;\theta), y_i) + \lambda G(\theta)\right\}. \tag{2.28}$$

Deep Learning

More recently, Abadi et al. [1] suggested an algorithm for privacy preservation with respect to every example within the dataset (this is based on gradient perturbation, see Algorithm 3). The main idea is basically to compute the gradients for every example within a group of size L (similar to a batch), and before averaging out all these gradients, they clip each one at a predefined threshold C. This way, they can impose an upper bound to the sensitivity of the algorithm. After averaging, authors add calibrated noise to it based on the clipping

threshold. This method is particularly attractive because the noise added this way, is inversely proportional to the group size (i.e., the noise is divided by the total number of examples within a group/batch). This way, they achieve a significantly less noise addition to the model's gradients (e.g., than adding noise without dividing by the group size L), and at the same time, they can provide very attractive privacy guarantees.

Authors experimented with simple Feed Forward Neural Networks, some of which had pre-trained Convolutional layers (in a transfer learning fashion) and showed, that overall this method is not damaging the utility significantly, as they achieved accuracies close to the non-differentially private state-of-the-art models.

Algorithm 3 Differentially private SGD [1].

Input: Examples $\{x_1, ..., x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $g_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip Gradient

$\bar{g}_t(x_i) \leftarrow g_t(x_i)/\max(1, \frac{||g_t(x_i)||_2}{C})$

Add Noise

$\tilde{g}_t \leftarrow \frac{1}{L}(\sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}))$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{g}_t$

Output θ_T and compute the overall privacy cost (ϵ, δ) using a privacy accounting method.

Other Applications

One can find examples of DP in Bayesian Learning, Matrix factorization & Clustering; Wang et al. [106] propose the one posterior sample mechanism (OPS) where they can protect privacy "for free" after noticing that sampling from the posterior with bounded log-likelihood, is basically the exponential mechanism as seen in [67], Dimitrakakis et al. explore the same idea in [18]. Park et al. in [79] demonstrate a DP version of Variational Bayes.

Dwork et al. in [27] describe a DP version of the Principal Components Analysis (PCA) algorithm for dimensionality reduction. They achieve this by perturbing the inputs (i.e., adding noise to covariance matrix $X^\top X$), and then, they compute the principal directions of the data using that noisy matrix instead. Nissim et al. in [76] show a version of the DP k-means algorithm for clustering using the sample and aggregate framework.

2.5 Related work

As seen earlier, in [92, 5, 1], authors rely heavily on applying repeatedly the additive noise mechanism on the gradients, at each SGD step. According to the basic composition theorem and the advanced versions of it [22, 26, 49, 25], the privacy loss after each step is going to increase. Privacy Loss definition allows us to quantify this degradation and also permits the analysis and control of cumulative privacy loss over multiple computations.

To this end, Abadi et al. propose in [1] the *Moments Accountant*. This is a way to calculate how much "privacy budget is being spent" at a time. Authors achieve this by treating the privacy loss as a random variable (rv), and then, they use a moment-generating function to calculate higher moments of this rv, which then they bound to prove a DP guarantee. This way, they can keep track of the cumulative loss at each step, so that the training process may be interrupted after some desired threshold value of ϵ or δ has been reached. In addition, to demonstrate this accountant, authors are using a variation of SGD, namely DP-SGD, similar to [92, 5] (i.e., gradient perturbation, Algorithm 3), to train deep neural networks for classification. Their experimental results indicate both high accuracy and low privacy loss, setting the current state-of-the-art in training models under privacy guarantees.

Within deep learning, Papernot et al. [77] propose a framework named PATE. They protect the privacy of the training set during learning by transferring knowledge from an ensemble of teacher models (trained on partitions of the private data) to a student model. More recently, authors expanded this work into [78] to show the potential of their system on bigger, real-world datasets.

Differential Privacy and Federated learning systems have received increasing attention lately as several attacks and defenses that undermine privacy in that context have been proposed. In [42], authors propose a Generative Adversarial Network (GAN) that is able to reconstruct (up to a certain degree) parts of private dataset, while in [69], authors demonstrate that periodically exchanging model updates can leak unintended information about participants' training data. Bhowmick et al. [7] suggest ways of training decentralized models, under strong DP guarantees, such that certain adversaries cannot reconstruct an individual's data.

In addition, we identify two very similar works, as seen in [33, 66]. In [66], authors learn DP language models using Recurrent Networks, while in [33], authors perform DP image recognition using Convolutional Networks. They both take advantage of the Federated Learning framework by [65] to distribute the learning task among N different clients. Recently, an extension of [66] has been proposed in [99] that examines the concept of adaptive clipping of clients' updates with the prospect to eliminate the need for extensive parameter tuning in federated settings.

These approaches can be characterized as a perturbation of the output of the optimization process (such as [110, 48]); Every client, learns a model (from its own local data partition) which is not differentially private per se. To ensure

differential privacy, authors average the local models to come up with a new one (under Federated optimization paradigm, see Algorithm 1), and then, calibrated noise is added to it. We note that in order to come up with a reasonably good model, a significant number of communication rounds between the participating clients and the server must be executed. This means that the privacy mechanism has to be applied over the course of the rounds. To account for this privacy loss, authors make use of the moments accountant by Abadi et al. [1] and stop the process when a certain threshold value of ϵ or δ is reached.

In both cases, the authors assume that the central aggregator who averages the client's models is a trusted entity. Their motivation is basically to achieve user-level privacy in a sense, that ensures, that the learned model does not reveal whether a client participated during decentralized training.

To this end, we build on existing methods and propose a new framework that can provide end-to-end privacy guarantees without assuming the existence of a trusted curator. The suggested framework is able to provide formal privacy guarantees for every single data-point the user might hold, and, at the same time, we show that with minor costs in utility, we can provide additional user-level privacy guarantees.

Chapter 3

Methodology

In this chapter, we will present the suggested framework along, with the datasets and models developed as a part of this thesis in greater detail.

3.1 Dataset

For the experimental part, we develop models based on the popular *MNIST* dataset [59]. The motivation behind selecting MNIST for the experiments stems from the fact that it is commonly used as a test-bed for new approaches and acts as a proof of concept for many works within ML-Privacy research community, thus the presented work can be easily compared to other state-of-the-art approaches. The dataset consists of 60,000 examples of labeled, handwritten digits (0-9) for training purposes and an additional 10,000 digits for evaluation.

3.2 The Models

Before presenting the final approach, which is a differentially private federated algorithm, we firstly provide details on the baseline models. These include non-differentially private Federated Learning models as well as Differentially Private models outside the federated optimization framework, which we eventually combine to provide the suggested method. In Chapter 4, we provide more implementation details and insights on the experiments we performed both from baseline models as well as the suggested method.

3.2.1 Federated Learning

Essentially, federated learning [65] (see Algorithm 1), can be seen as a form of fine-tuning a central model on clients' local datasets. This is achieved as each client contributes its updates Δw, that is, $w = w - \eta \nabla \ell(w; b)$, to the central model. The central model, then, is basically performing a weighted average on all participating clients' models such as, $w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ to come up with the new model (n_k corresponds to k^{th} client's number of datapoints it holds). The number of updates that each client contributes to the global model (per communication round) is based on hyper-parameters that we can tune. For example, if the local dataset is made of 600 data points, and we decide to split this

into batches of size 60 (i.e., 10 batches in total) and set the number of epochs to two, then, the client will contribute with a gradient based on 20 updates in total (10 batches $\times$ 2 epochs). Therefore, the workload can be fully controlled by the number of local epochs, the batch size, and the number of clients participating in each round.

Since we are dealing with a multi-class classification problem, we train the local models under the cross-entropy loss function, and, at the output, we use a Softmax activation function that gives us a probability distribution over the predicted classes. We consider two models; a standard Multilayer Perceptron and a Convolutional Neural Network. In addition to this we consider two data partitions; one with **i.i.d.** samples and one with **non-i.i.d.** samples.

- **I.I.D.** samples. For the i.i.d. case, the dataset is shuffled, and it is being split into evenly sized data chunks which, then, are assigned to participating clients and act as their local dataset.
- **non-I.I.D.** samples. This is more of a real-life scenario as clients may not hold items that belong to the same classes. That is, one client might hold digits from classes 1 and 2 while another client might hold digits from classes 2 and 9. In this case, the dataset is partitioned such that each client holds about two digits in its local dataset.

3.2.2 Differentially Private Model

This is a model based on Abadi et al. [1] (see Algorithm 3), where they guarantee that the model is differentially private with respect to every data point within the dataset. We create this as a baseline model to test the per-example data privacy. As we mentioned earlier (see Chapter 2), to realize this model, we need access to the gradient of each individual example. Unfortunately, most of the current automatic differentiation tools commonly used for machine learning and/or deep learning purposes do not support this functionality. However, they do compute the gradient of the mini-batches.

One way to achieve this, is, by naively setting the mini-batch size to one in order to obtain the gradient of each example. This is a rather intractable approach, especially when dealing with modern datasets, which usually contain a large number of examples. To this end, we implement an efficient way of calculating each gradient of the examples within a mini-batch using the approach suggested by Goodfellow in [34]. This only adds up a minor computational overhead compared to the naive method.

It is worth noting that due to the complexity of the convolutional layers, in particular, due to the fact that they share parameters (see Chapter 2), there is not yet an efficient way to calculate per-example gradients for this type of deep neural networks. The per-example models that we develop, therefore, are based on simple multi-layer perceptron (MLP) architectures. Finally, to calculate the privacy spending at each step, we make calls to the Moments Accountant API[1], as seen in [1].

[1] https://github.com/tensorflow/privacy/blob/master/privacy/analysis/rdp_accountant.py

3.2.3 End-to-end DP Federated Model

This is a fusion model where each client runs the Differentially private SGD algorithm (inspired by Algorithm 3), and the server is aggregating the local models using Algorithm 1. Due to the limitations concerning the per-example gradients computations that we discussed earlier, each client's model consists of an MLP that minimizes the cross-entropy loss with a softmax activation function in the output layer. We consider both iid and non-iid dataset distributions for the experimental part.

This method is able to provide end-to-end privacy guarantees. That is, by protecting each example's privacy in the local privacy setting (i.e., each client has a DP model), we show in the following part that this has certain global privacy implications.

3.3 Privacy Analysis

Introduction

In order to provide formal privacy guarantees in terms of (ϵ, δ)-differential privacy for the proposed algorithm, we need to accurately count the privacy loss. Privacy loss occurs at every step where there is access to data (for example, when we use the privacy mechanism to approximate the average gradient during the DP-SGD steps). A way to quantify this loss, for example, is by using the strong composition theorem [26], however here we make use of the Moments Accountant, proposed in [1] since it provides a much tighter estimation of the privacy loss.

Privacy accounting is a procedure that can be either executed offline, for example, before the training part, so that we know in advance for how long can we train a model given some DP constraints, or online and stop the training phase once an upper limit for ϵ or δ has been reached. For the accounting part, mainly three variables need to be considered; 1) The number of steps, T, that the Gaussian Mechanism is applied on the model, 2) the Sampling Rate, q, and 3) the standard deviation, σ, of the additive Gaussian Noise. Firstly, we review the per-example privacy, and, following this, we provide the per-client privacy analysis.

Suppose a partition P of dataset $\mathcal{D}$, such that $P = \{d_1, d_2, ...d_n\}$ and $\mathcal{D} = \bigcup_{i=1}^{n} d_i$, where n is the total number of clients, such that each client is assigned with a *disjoint* subset, d_i, from $\mathcal{D}$. Each d_i consists of z samples such that $d_i = \{s_1, s_2, ...s_z\}$ (and $\mathcal{D}$ consists of $n \cdot z$ samples in total).

Per Example Privacy

In the per example privacy, we are interested in protecting each client's data points. That basically means that this way we can provide under some formal $(\epsilon, \delta)-$DP guarantee that the client's i, learnt model will not reveal whether any single data-point, $s \in d_i$ they possess, was used during the training procedure or not.

The number of steps, T, refers to how many times we apply the Gaussian Mechanism during the training phase. In the simplest case, where no clients are involved (i.e., simple DP-SGD), we need to count how many times have we stepped towards the steepest direction of the gradient. For example, if during one epoch, the algorithm goes through 10 mini-batches and, the training phase consists of 100 epochs, then we count $T = 1000$ steps, in total. In the Federated Learning setting, the clients come in rounds. That is, a client may be picked multiple times throughout the training phase. Therefore, if some client has performed $T = 100$ steps during the first round, and then picked again, at a later time during the training phase, its total steps are increased per every round, whether it is participating in it or not (since there is a non-zero probability of the client's participation). Thus, the total steps in a federating setting for the clients can be calculated as follows: if b, denotes the batches per epoch, cr denotes the communication rounds, and ep the number of epochs, then steps $T = b \cdot cr \cdot ep$.

The sampling rate, q, has to do with the portion of samples from the dataset $\mathcal{D}$, being used during each step of the gradient descent algorithm. For example, in the simple case, if during an epoch the algorithm is using l examples at every SGD step (i.e., l = mini-batch size), then $q = \frac{l}{n \cdot z}$. In the Federated setting, each client i, has its own mini-dataset d_i of size z, however, only a portion of these clients are active at a given round. That is, we need to take into consideration only their contribution of data while training. Thus, if during a communication round a number of p clients out of n are participating and during an epoch each one is using l_c examples at every local-SGD step, then $q = \frac{p \cdot l_c}{n \cdot z}$.

Finally, regarding the standard deviation, σ, it can be treated as a hyper-parameter that can be tuned using random search, for example. Typically, a higher σ will add more noise, which will be better for privacy. However, too much of this quantity may significantly deteriorate the accuracy of the learnt model.

Per Client Privacy

In the per-client privacy, we are interested in protecting the client per se. That is, we would like to provide plausible deniability for its participation in the training procedure. We suggest that training clients with per-example privacy can additionally provide a sufficient $(\epsilon, \delta)-$dp guarantee for per-client privacy at a minor cost in client's additive noise mechanism. We show that the aggregated model inherits a Gaussian Mechanism from the per-example privacy of the clients with a certain sensitivity S_f and standard deviation σ.

In this case, at each communication round, we treat each participating client's trained model as a "noisy black-box model", which is then passed (along with the other clients' models) to the model aggregator to calculate the central model. Therefore, a Gaussian Mechanism is applied at the aggregator's side every time it is averaging out the noisy clients' models, that is, once per communication round. Thus, the number of steps, T, (that the GM is applied) in this case is equivalent to the number of communication rounds irrespective of the number

of participating clients.

Regarding the sampling rate, since we are interested in hiding the clients' participation, we refer to sampling rate q, as the fraction of active clients p that we randomly sample from the clients' pool at each round. That is $q = \frac{p}{n}$.

Finally, regarding the standard deviation, σ, that is applied at each communication round on the inherent Gaussian Mechanism of the aggregator we show that we can recover a calibrated standard deviation $\sigma_{cal} = \frac{\sigma}{\sqrt{k}}$ and sensitivity $S_f = k \cdot S$. Where σ is the original standard deviation we used in the per-example training for the clients, k is the number of SGD steps that a client takes and, S stands for the original sensitivity of the clients.

Recounting Noise

Theorem 1. *Let a model $\mathcal{R}$, created as an average of DP-models using DP-SGD with k steps per DP-model. Then, $\mathcal{R}$, is also differentially private via an Inherited Gaussian Mechanism, parameterized by sensitivity $S_f = k \cdot S$, and standard deviation $\sigma_{cal} = \frac{\sigma}{\sqrt{k}}$, where σ and S is the original standard deviation and sensitivity respectively, used during the DP-SGD training for the DP-models.*

Proof. We would like to know how much of the per-example noise injected during the training phase of the participating clients can be re-claimed for per-client privacy accounting.

After training n clients at some communication round, the server averages out the clients' local models such that:

$$\text{server\_model} = \frac{1}{n} \sum_{i=1}^{n} cl_i \tag{3.1}$$

.
where cl_i the i^{th} client. We can re-write 3.1 as:

$$\begin{aligned} \text{server\_model} &= \frac{1}{n} \sum_{i=1}^{n} (\theta_i + \mathcal{N}(0, S_i^2 \sigma_i^2)) \\ &= \frac{1}{n} \sum_{i=1}^{n} \theta_i + \frac{1}{n} \sum_{i=1}^{n} \mathcal{N}(0, S_i^2 \sigma_i^2) \end{aligned} \tag{3.2}$$

where θ_i is the client's i local model and $S_i^2 \sigma_i^2$ is the variance. Since each client gets trained with the same amount of noise at each communication round (i.e., S_i^2 and σ_i^2 are identical $\forall cl_i$), the noise distribution of the averaged model is governed by $S^2 \sigma^2$.

Therefore, in order to analyze the effective noise that can be re-used by the server to preserve client-based differential privacy it is sufficient to study a single client from a "total noise" perspective.

We can calculate the total variance of a client as:

$$\text{Var} = k \cdot (S^2\sigma^2) \tag{3.3}$$

where k, accounts for how many times (steps) the Gaussian mechanism has been applied to the client. Therefore, the total (uncalibrated) stdev, σ_{uc}, is:

$$\sigma_{uc} = \sqrt{k \cdot (S^2\sigma^2)} \tag{3.4}$$

In order to be able to make claim for this noise we need it to be calibrated according to the f's total sensitivity. For the per-example privacy, we clip every example's gradient g that its norm is $>$ than some threshold S such that it gets scaled down to the norm of S. Then, after the clipping procedure, all these gradients are being averaged out.

Therefore, for these gradients g, we know that: $\forall i \in \{0, \ldots, m\},\ ||g_i|| \leq S$. We can trivially show, using the triangle inequality and the absolute homogeneity, that the average gradient is bounded by S:

$$\begin{aligned} ||\frac{1}{m}\sum_{i=1}^{m} g_i|| &\leq \frac{1}{m}\sum_{i=1}^{m} ||g_i|| \\ &\leq S. \end{aligned} \tag{3.5}$$

At every step k, the algorithm is moving towards some minima using SGD. This means, that, the average gradients, $\tilde{g}$, from each mini-batch are being summed. Generalizing the above Equation:

$$\begin{aligned} ||\sum_{i=1}^{k} \tilde{g}_i|| &\leq \sum_{i=1}^{k} ||\tilde{g}_i|| \\ &\leq k \cdot S \end{aligned} \tag{3.6}$$

which gives us an upper bound for the total sensitivity S_f, with k accounting for the total number of steps of SGD.

The Gaussian Mechanism is defined as:

$$\mathcal{M}(d) \triangleq f(d) + \mathcal{N}(0, S_f^2 \cdot \sigma^2) \tag{3.7}$$

.

Therefore we can recover the calibrated σ_{cal} as:

$$\begin{aligned} \text{Var} = S_f^2 \cdot \sigma_{cal}^2 &\Leftrightarrow \\ \sigma_{cal} &= \frac{\sqrt{\text{Var}}}{S_f} \\ &= \frac{\sqrt{k \cdot (S^2\sigma^2)}}{k \cdot S} \\ &= \frac{\sqrt{k} \cdot \sigma}{k} \\ &= \frac{\sigma}{\sqrt{k}}. \end{aligned} \tag{3.8}$$

where σ, is the client's original stdev. □

3.4 Pseudo-code

In this section, we provide the proposed algorithm in pseudo-code for the end-to-end differential privacy, within federated learning, as well as the method for privacy accounting.

3.4.1 Proposed Algorithm

Algorithm 4 Proposed Method for End-to-End Differential Privacy.

Input: Examples $\{x_1, ..., x_N\}$, loss function $\mathcal{L}(w) = \frac{1}{N}\sum_i \mathcal{L}(w, x_i)$. Parameters: learning rate η_t, noise scale σ, batch size B, gradient norm bound S, local epochs E, target epsilon per example ϵ_{ex}, target epsilon per client ϵ_{cl}, total clients K, fraction of participating clients per round C and $\mathcal{P}_k$ is the local data partition of client k with size $|\mathcal{P}_k|$.

Server Executes:
initialize w_0 randomly
$\sigma_{rec} \leftarrow \frac{\sigma}{\sqrt{\frac{|\mathcal{P}_k|}{B} \cdot E}}$
$m_t \leftarrow \max(C \cdot K, 1)$
for each round t =1,2,... **do**
  $Z_t \leftarrow$ (sample without replacement a set of m_t clients from K)
  for each client $k \in Z_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
  print privacy_spent(σ_{rec}, t, parameters)

ClientUpdate(k, w_t): //Runs on Client k
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches b of size B)
for $t \in E$ **do**
  for batch $b_t \in \mathcal{B}$ **do**
    Compute gradient
    For each $i \in b_t$, compute $g_t(x_i) \leftarrow \nabla_{w_t}\mathcal{L}(w_t, x_i)$
    Clip Gradient
    $\bar{g}_t(x_i) \leftarrow g_t(x_i)/\max(1, \frac{||g_t(x_i)||_2}{S})$
    Add Noise
    $\tilde{g}_t \leftarrow \frac{1}{B}(\sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 S^2 \mathbf{I}))$
    Descent
    $w_{t+1} \leftarrow w_t - \eta_t \tilde{g}_t$
return w_{t+1}

3.4.2 Privacy accounting

Algorithm 5 Privacy Accounting.

Input: Moments accountant $\mathcal{M}$.

privacy_spent(σ_{rec}, t, **parameters**):

// Per Example Accounting:
Per-example Sampling rate (q_{ex})**:**
$q_{ex} \leftarrow \frac{B}{|\mathcal{P}_k|} \cdot C$
Per-example steps (st_{ex})**:**
$st_{ex} \leftarrow \frac{|\mathcal{P}_k|}{B} \cdot E \cdot t$
$\delta_{ex} = \mathcal{M}(\sigma, st_{ex}, q_{ex}, \epsilon_{ex})$

// Per Client Accounting:
Per-client Sampling rate (q_{cl})**:**
$q_{cl} \leftarrow C$
Per-client steps (st_{cl})**:**
$st_{cl} \leftarrow t$
$\delta_{cl} = \mathcal{M}(\sigma_{rec}, st_{cl}, q_{cl}, \epsilon_{cl})$

return δ_{ex}, δ_{cl}

3.5 Implementation

The implementation part of this thesis (Algorithms and Scripts 1, 3, 4, 5, 6, 7 and 8) was written in Python [104]. In particular, the Neural Network models, were built in Pytorch [80] framework for Deep Learning. The experiments were conducted at EPFL's IC Cluster which contains dedicated Nvidia CUDA-GPU instances required for the graphs' computations.

Chapter 4

Experiments & Results

In this chapter, we introduce more details about the experimental procedure, and, in addition to this, we present our results. On top of that, we try to shed some light on how the hyper-parameters affect the overall performance and the related trade-offs one should consider when training models under DP guarantees.

4.1 Dataset Partitions

We consider two data partitions, one with iid examples (i.e., we shuffle the dataset) and one with non-iid examples. In the latter case, we do it in accordance to [65], that is, the dataset is first sorted by digit labels, we divide this into 200 shards of size 300, and then we assign to each of the clients two shards (i.e., we consider in total one hundred clients in case of MNIST dataset and each one gets 600 examples in total). The non-iid case is a more realistic scenario (from a federated learning point of view) as we don't assume the same class distribution for every client. For example, data points could represent songs or playlists of some specific genre. Therefore, some clients may share some genres, but they cannot have the same genre distribution (as in iid case).

Apart from one hundred clients, we examine cases with more clients. Specifically, we perform experiments with one thousand and ten thousand clients. For these experiments, we repeat the data points to accommodate additional clients.

4.2 Model Details

We perform experiments based on two different models. To test the federated learning algorithm baseline, we train a Convolutional Neural network with two 5×5 convolution layers; the first with 32 channels, the second with 64, each followed by 2×2 max pooling, a fully connected layer with 512 units and ReLu activation, and a final softmax output layer (this model is the same presented in [65]). It yields in total 1,663,370 parameters. We also consider a Multi-Layer Perceptron (MLP) that consists of one hidden layer with 1000 units followed by a ReLu activation function and an output layer with a softmax activation function. This model yields in total 795,010 free parameters. For the differentially private

case, we consider only the MLP for reasons explained in Chapter 3, section 3.2.2. Both models are trained to minimize the cross-entropy loss.

4.3 Differentially Private Algorithms

We study two additional algorithms in order to perform experiments and, to compare their performance and DP guarantees with the proposed method. Firstly, we consider an algorithm that is based on the trusted curator framework. In this case, each client trains on its local data partition, and then, the resulting (non-private) local models are being sent to the trusted curator to average them out, add noise, and then, return the average model to the clients (see also Related work, section 2.5).

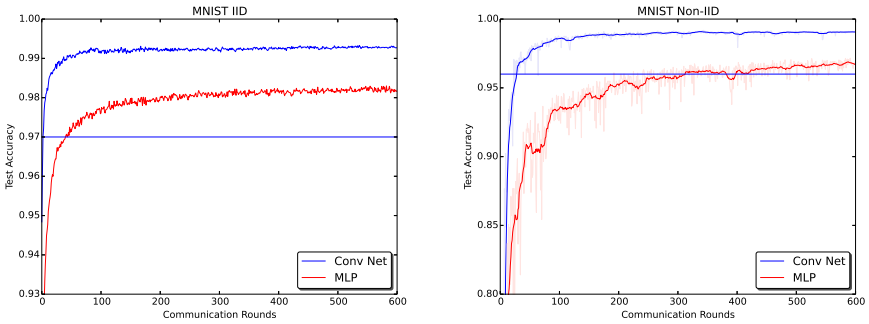
This algorithm is inspired by [33], however, due to the fact that authors clip the contributions of each client by the median of norms of their updates, which constitutes a privacy violation, instead of this, we use a constant for the clipping part. Finally, we consider a naive version for end-to-end differential privacy within federated learning.

For the naive case, each client performs computations for per-example privacy as in the suggested method, but, in addition to this, we add more noise at the averaging phase (i.e., we treat the averaging phase like the trusted curator framework). The proposed method consists of Algorithms 4 and 5 and can be found in section 3.4. In addition, in the Appendix A, we have attached in pseudo-code a collection for all the dp-federated algorithms implemented in this thesis; The trusted curator (server-side noise) method corresponds to Algorithm 7, while, the naive version corresponds to Algorithm 8.

4.3.1 Baseline results

Federated Learning

For this experiment, we set 100 clients in total, where 10 of them are randomly sampled in each communication round in order to participate. Each client performs 10 local epochs on its data with the batch size set to 10 and, the local data partition is set to 600 data points in total. The learning rate is set to 0.03. In Figure 4.1a and Figure 4.1b we illustrate the results for the iid and non-iid cases respectively.



(a) Accuracy with IID setting.

(b) Accuracy with the pathological Non-IID setting.

Figure 4.1: Test set accuracy within Federated Learning. The results in these plots are not differentially private, they merely give us an idea of the test set performance of the non-private setting.

We notice that, in general, the Convolutional Neural Network (CNN) performs better, and in addition to this, converges much faster compared to the MLP. This is especially true in the pathological non-iid case where the MLP struggles with the non-identical class distribution, whereas, the CNN even reaches the accuracy levels of the iid setting in merely a hundred of communication rounds. This result comes as no surprise since CNNs are known for the rich feature representations they extract from the inputs, which ultimately leads to better and more stable performance. The summary can be seen in Table 4.1.

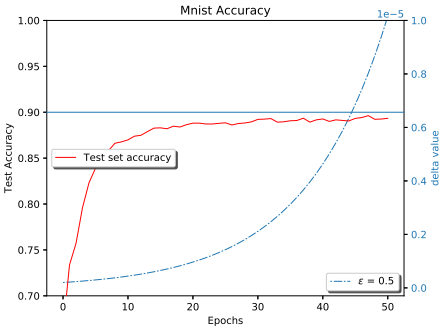
Model	IID (%)	non-IID (%)	Comm. Rounds
CNN	**99.33**	**99.18**	600
MLP	98.30	97.16	600

Table 4.1: MNIST Accuracy in Federated Learning.

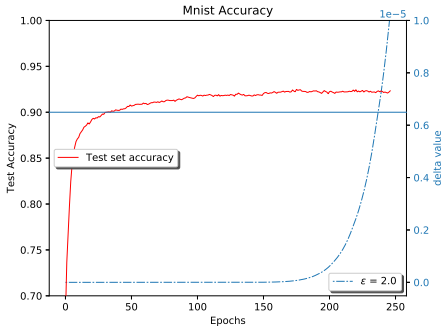
Differentially Private Model

In this setting, we test the performance of the per-example privacy, outside of the federated learning framework with iid samples. We train the MLP for as many epochs as possible until an acceptable predefined upper limit of (ϵ, δ)-dp guarantee has been reached.

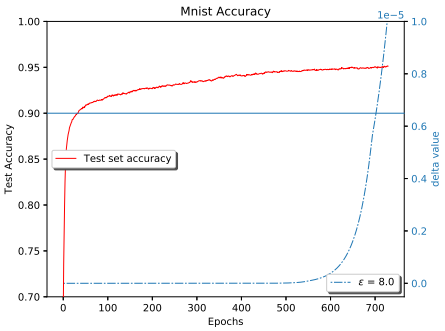
We set the batch size to 600, the clipping threshold to 4 and, the learning rate to 0.02. We set the lowest accepted δ value for these runs to 1e-5 (once this value has been reached training procedure is stopped). We try a combination of different σ and ϵ values. The results are illustrated in Figure 4.2.



(a) Large noise, $\sigma = 8.0$.



(b) Medium noise, $\sigma = 4.0$.



(c) Small noise, $\sigma = 2.0$.

Figure 4.2: Test set accuracy for Differentially Private models on MNIST dataset. The blue axis with the dashed line, corresponds to the delta value at an epoch, given some predefined ϵ privacy budget.

The general trend is that the more we relax the ϵ requirements, the more epochs we are allowed to run the algorithm. As a result of this, better accuracy is achieved. The ϵ is basically controlling the privacy leakage, and as we relax this constraint, we may add significantly lower amounts of calibrated noise on the mechanism compared to some very strong ϵ. For example, in Figure 4.2a , we see that for a very strong privacy guarantee of $\epsilon = 0.5$ we are only allowed to run 51 epochs at very high noise levels of $\sigma = 8.0$ until δ (the probability that mechanism fails) reaches the predefined threshold of 1e-5 in an exponentially fast manner. The summary can be seen in Table 4.2.

σ	$(\epsilon, \delta)-$dp	Epochs	Accuracy (%)
8.0	(0.5, 1e-5)	51	89.63
4.0	(2.0, 1e-5)	247	92.49
2.0	(8.0, 1e-5)	730	95.16

Table 4.2: MNIST Accuracy in (per-example) Differential Privacy.

These results can be further improved; In [1], authors explore the idea of performing differentially private PCA on the data before learning the neural model,

which as they report, can yield up to a 2% of constant increase over this baseline. However, we note that accounting for every dp-PCA step equates to less training epochs for the neural model since the privacy budget is consumed faster.

4.3.2 Differentially Private Federated Algorithms

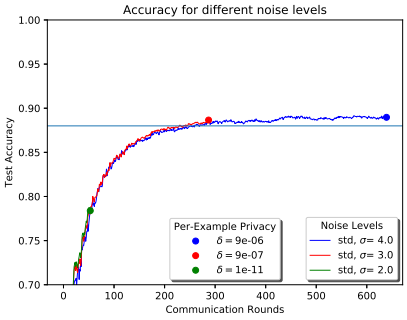
In this section, we present the results of the proposed method (Algorithm 4) and compare it to the other two algorithms, briefly introduced in section 4.3 (Algorithms 7 and 8).

Hyper-parameter setup

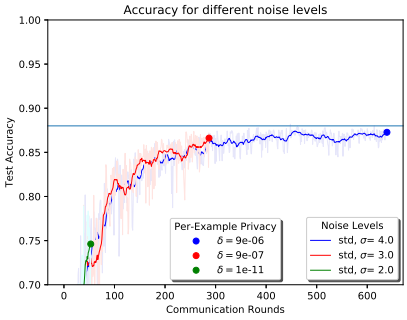
We performed experiments with variable client size (100, 1000, and 10000) both for iid and non-iid class distributions. Each client's local dataset partition consists of 600 data points, and the batch size is set to 100 samples. In addition, the clients perform only one epoch on their dataset at every communication round they participate in. This way, each client is assigned with a very small computational overhead, which translates into a contribution of merely six model updates per communication round. We performed a low-resolution grid search for the hyper-parameters and set the learning rate to 0.02 across all algorithms, while the clipping threshold is set to 0.4 for the trusted curator algorithm. For the proposed method, we employ a non-linear way to decay the clipping threshold throughout the communication rounds.

Proposed Algorithm

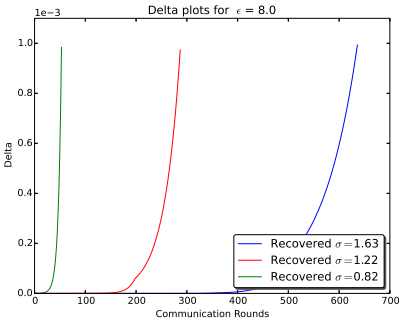
In Figure 4.3, we look into how the different noise levels affect the performance both in terms of privacy and utility for 100 clients and participation rate per round $q = 0.1$ (i.e., 10 clients participate per round). For this experiment, we stop the training part, when either the per-example privacy or the per-client privacy δ reaches $1e-5$ or $1e-3$ respectively. The ϵ for per-example privacy is set to 1.31, which constitutes a relatively strong guarantee, while, for the per-client, we set it to the more relaxed value of 8.0. As a rule of thumb, one may choose $\delta < \frac{1}{N}$, where N corresponds to the number of data points (or clients in the case of per-client privacy). So, in this case with 100 clients, for the per-example privacy part, we are basically trying to "hide" each data point of each client with respect to all clients' data points (i.e., $N = 100 \times 600$, where 600 is the local dataset size allocated to each client). In the following experiments, we aim for δ to be *about* an order of magnitude smaller than $\frac{1}{N}$.



(a) Accuracy with IID setting.



(b) Accuracy with the pathological Non-IID setting.



(c) Per-Client privacy.

Figure 4.3: Test set accuracy and privacy guarantees of the proposed method with 100 clients and variable σ values, (a) corresponds to the IID setting while (b) to the Non-IID setting. In (c) we notice the effect of the recovered σ on clients' δ values and how it limits the training, especially in the cases where the recovered σ is rather small.

We notice that smaller amounts of σ on per-example privacy, tend to recover even smaller amounts of σ for accounting per-client privacy (see Privacy Analysis in section 3.3). Ultimately, this doesn't allow us to run more epochs, as the upper (per-client) privacy limit has been reached (see Figure 4.3c). In particular, the main issue is that although with $\sigma = 2.0$, we can achieve an impressive δ for the per-example part, this still doesn't allow us to run more epochs since δ from per-client part grows exponentially fast and stops the training. This issue becomes smaller as the number of clients is increased, and more communication rounds can be realized. The summary can be seen in Table 4.3.

Effect of Clipping Threshold

The role of gradient clipping in machine learning is mainly to suppress gradients with high magnitude, which may lead to a phenomenon often referred to as the *exploding gradients* [35]. This is commonly observed in recurrent neural networks and basically hinders the optimization procedure from converging to the correct minimum landscape on the loss surface. Within differential privacy,

σ	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Epochs	Accuracy (%)
4.0	(1.31, 9e-06)	(8.0, 1e-3)	640	89.45
3.0	(1.31, 9e-07)	(8.0, 1e-3)	288	88.66
2.0	(1.31, 1e-11)	(8.0, 1e-3)	54	78.73

(a) IID

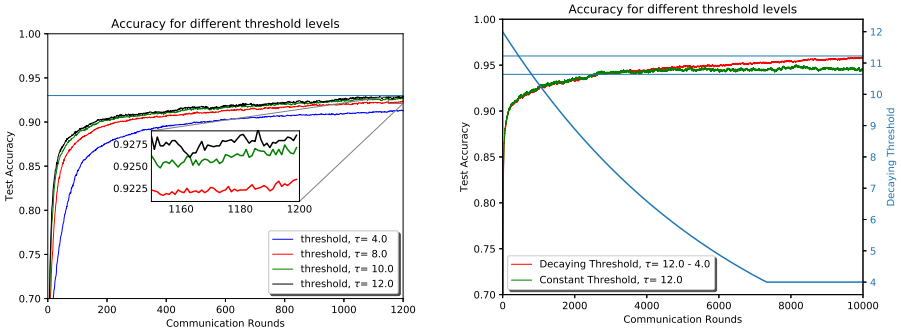
σ	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Epochs	Accuracy (%)
4.0	(1.31, 9e-06)	(8.0, 1e-3)	640	88.19
3.0	(1.31, 9e-07)	(8.0, 1e-3)	288	87.37
2.0	(1.31, 1e-11)	(8.0, 1e-3)	54	74.62

(b) non-IID

Table 4.3: IID & non-IID performance for 100 clients with the proposed method.

the clipping threshold is an important parameter as we can use it to impose the sensitivity of the function we would like to make private.

Setting a clipping threshold will allow gradients with a magnitude lower than it un-clipped. In other words, setting a high threshold would allow most of the gradients un-clipped, which means that the batch gradient will remain unbiased. However, this way, more noise is added in total as the clipping threshold is multiplied by the standard deviation, σ, to compensate for this. On the other hand, if we impose a very small clipping threshold, this would allow less noise into the



(a) Fixed Thresholds and accuracy over a small number of communication rounds.

(b) Threshold decay and accuracy over an extended number of communication rounds.

Figure 4.4: Clipping threshold and its effect on accuracy. The blue axis in (b) shows the non-linear way of threshold decay we employ.

system. However, the bias introduced by this action on the gradients can turn the model untrainable. Therefore, finding a good trade-off between adding too much noise and biasing the gradients through this threshold is of high importance.

To address this issue, we perform experiments with variable clipping thresholds (see Figure 4.4). As we can see in 4.4a, during the early communication

rounds, applying a rather large clipping threshold can increase the performance considerably. However, in the long run, keeping the threshold constant at a high value doesn't add much to the utility. Instead, we notice that starting at a large clipping threshold value and by decaying it in a non-linear way (see Figure 4.4b and code snippet 6) we can benefit in utility by as much as 2%.

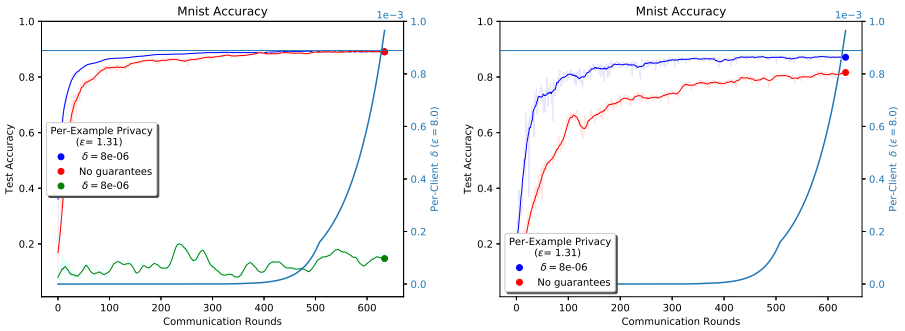
The reason for this, is, that when we are at an early stage of the optimization procedure, it is more important to get as many unbiased gradients as possible in order to find the correct slope on the loss surface that minimizes the cost function while at the later stages where the process is converging to the true minimum is more important to avoid the noise induced by the clipping threshold in order to find more fine-tuned minimum regions on the loss surface.

Comparative Results

In addition to the experimental setup introduced earlier, we note that in the case of naive and server-side noise algorithms, the σ value we use for the per-client privacy protection is the same as the σ we recover from the proposed method. This way, all algorithms are compared fairly, over the same course of communication rounds.

More specifically, for 10^2 and 10^3 clients, we use $\sigma = 4.0$ which recovers $\sigma_{rec} = 1.6329$. For 10^4 clients, we use $\sigma = 1.85$ which recovers $\sigma_{rec} = 0.7552$. The client participation rate per round is set to 0.1, 0.05 and 0.005 respectively. In the plots below, the blue color corresponds to the proposed method, the red color corresponds to the trusted curator method, while with green color, we show results on the naive method. We note that since the results from the naive method are not very encouraging, we run experiments only for the iid label distribution in that setting.

100 Clients



(a) IID case.

(b) non-IID case.

Figure 4.5: 100 Clients. Blue curve corresponds to the proposed method, red curve corresponds to the trusted curator method, while the green curve corresponds to the naive method. Light blue axis shows the δ value progress per epoch for per-client privacy.

Algorithm	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Rounds	Accuracy (%)
Proposed Method	(1.31, 8e-06)	(8.0, 1e-3)	635	**89.08**
Trusted Curator	*No Guarantees*	(8.0, 1e-3)	635	88.87
Naive Method	(1.31, 8e-06)	(8.0, 1e-3)	635	14.59

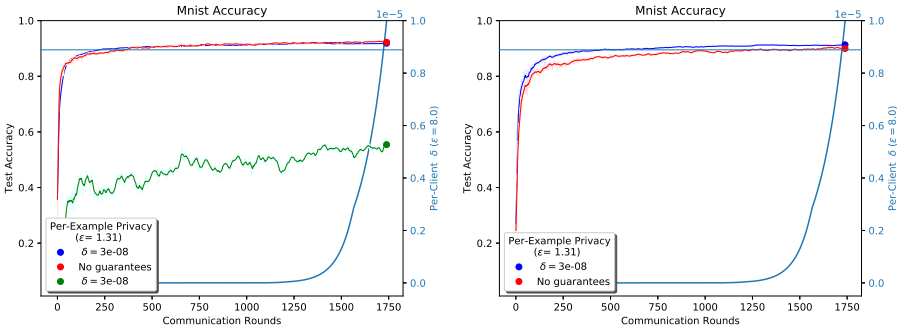
(a) IID

Algorithm	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Rounds	Accuracy (%)
Proposed Method	(1.31, 8e-06)	(8.0, 1e-3)	635	**87.08**
Trusted Curator	*No Guarantees*	(8.0, 1e-3)	635	81.63

(b) non-IID

Table 4.4: Comparison of IID and non-IID performance for 100 clients.

1,000 Clients



(a) IID case. (b) non-IID case.

Figure 4.6: 1,000 Clients.

Algorithm	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Rounds	Accuracy (%)
Proposed Method	(1.31, 3e-08)	(8.0, 1e-5)	1740	91.86
Trusted Curator	*No Guarantees*	(8.0, 1e-5)	1740	**92.11**
Naive Method	(1.31, 3e-08)	(8.0, 1e-5)	1740	54.86

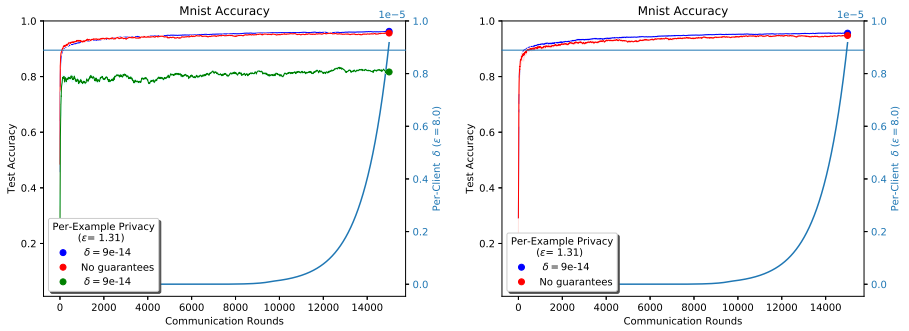
(a) IID

Algorithm	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Rounds	Accuracy (%)
Proposed Method	(1.31, 3e-08)	(8.0, 1e-5)	1740	**91.34**
Trusted Curator	*No Guarantees*	(8.0, 1e-5)	1740	90.12

(b) non-IID

Table 4.5: Comparison of IID and non-IID performance for 1,000 clients.

10,000 Clients



(a) IID case.

(b) non-IID case.

Figure 4.7: 10,000 Clients.

Algorithm	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Rounds	Accuracy (%)
Proposed Method	(1.31, 9e-14)	(8.0, 1e-5)	15000	**96.29**
Trusted Curator	*No Guarantees*	(8.0, 1e-5)	15000	95.64
Naive Method	(1.31, 9e-14)	(8.0, 1e-5)	15000	81.60

(a) IID

Algorithm	$(\epsilon, \delta)-$per example	$(\epsilon, \delta)-$per client	Rounds	Accuracy (%)
Proposed Method	(1.31, 9e-14)	(8.0, 1e-5)	15000	**95.73**
Trusted Curator	*No Guarantees*	(8.0, 1e-5)	15000	94.78

(b) non-IID

Table 4.6: Comparison of IID and non-IID performance for 10,000 clients.

As expected, the naive method cannot cope very well with all the additional noise needed to provide the per-example and per-client guarantees. Especially with 100 clients (Figure 4.5a), we notice that its results are bouncing around 14-15%, which is marginally above guessing rate for a 10-class classification problem. While more clients are participating (Figure 4.6a & 4.7a), the naive algorithm manages to overcome the guessing problem as the noise basically gets scaled down by the additional participants per round. However, its performance is still sub-par, especially when compared to the other two methods.

Regarding the proposed method and the trusted curator, we notice that most of the time, their performance (from utility perspective) is comparable, especially for the iid label distribution case. This doesn't come as a surprise, at least from a theoretical point of view, as the total (system-wide) noise σ, of both methods, is basically the same.

As a general trend, we notice that the proposed algorithm (especially in the

non-IID cases), behaves considerably better (less bouncing on the predictions). This is probably because the noise addition is part of the optimization process (i.e., noise is added directly to the gradients, and then we take a step towards the noisy minimum, we are basically learning noise). On the other hand, in the trusted curator framework, we learn a model, and noise is added to the learnt parameters after the learning process has finished, which leads to this unstable behavior, especially during the early runs (Figure 4.5b).

We note that the behavior of the trusted curator (red curve), in Figure 4.5b, arguably, with a higher resolution hyper-parameter tuning could bring it closer to the proposed method (blue curve), however, it is important that the proposed method does not deviate significantly from the trusted curator framework overall.

To sum up, in addition to strong per-example privacy guarantees, the proposed method provides reasonable guarantees at a per-client level with minimal loss of utility. That being said, the research question (see section 1.1), has been addressed in its entirety. Finally, we note that in real-world scenarios (with non-iid data distributions), where even more clients participate (than the experiments presented in this thesis), and vast amounts of data are involved, it is expected to perform at the same level as a non-DP algorithm since the additive noise required by the Gaussian Mechanism gets scaled down to the point that it shouldn't affect the utility.

Chapter 5

Conclusions

In this chapter, we draw the conclusions of this thesis. In addition, we discuss limitations and future work.

Conclusion

This thesis explored the idea of differential privacy within federated learning setting from a holistic point of view. That is, it tries to address the main limitation of the current framework (trusted curator model [33, 66]), which is the one-sided privacy guarantees offered by the trusted curator. It is shown both theoretically and experimentally that local privacy has global consequences. That is, end-to-end differential privacy can be realized within the proposed method, while the utility remains high and comparable to the state-of-the-art.

The proposed algorithm can scale down σ directly in proportion to the local mini-batch size of each client, and indirectly by the number of clients in the system and the dataset size. Roughly speaking, this gives this algorithm the advantage of injecting noise directly into the gradients of the clients' local models (as a part of their training procedure), which has two immediate effects. Firstly, it provides strong privacy guarantees at a data-point level, and, secondly, it allows us to recount this noise to provide an additional layer of privacy guarantees at a client level *without* explicitly adding more noise to the system.

The empirical results show that our proposed method can achieve similar, or better results compared to the trusted curator framework plus that additional level of privacy. It is also worth noting that since the noise injections are part of the training procedure, the overall learning curves can be significantly smoother and at the same time higher accuracies can be realized; This is especially evident in the non-iid case, which is the most realistic scenario for federated learning applications.

From a theoretical point of view, by recounting the noise, σ_{rec}, of the locally differentially private models of the clients (in order to find the system-wide noise), one would expect a very similar performance to the trusted curator model if $\sigma_{\text{curator}} = \sigma_{rec}$. This seems to be the case as the empirical results confirm this claim. However, we note that since additive noise is now part of the optimization

process, in practice, a much smoother behavior and relatively better accuracies are realized.

Limitations

At the time of writing this report, the main limitation lies within the current auto-differentiation tools (such as Pytorch, Tensorflow, etc.) as they don't support (efficient) access to the gradients of each individual example in the mini-batch. A bypass to this problem is the method suggested in [34], which is also used in this thesis. However, it is restricted to MLPs mainly. An efficient way for extracting individual gradients from Convolutional networks and Recurrent networks would enable a wider range of models to take advantage of this method.

Future work

There are several directions to be explored from this point. From an accuracy point of view, there are ways that can help it improve without lowering the σ and thus compromising the privacy guarantees; For example, doing dp-PCA (shown in [1]) could further improve the results. Although, for the time being, efficient per-example gradients for Convolutional networks are not available, it is widely shown that their features are transferable [19, 87] and so it is worth looking into this aspect within federated learning.

Another thing that is worth looking into is that when the dataset and the clients grow a lot, we end up having very strong privacy guarantees for the per-example part (which is expected). At the same time, the σ_{rec} we pick up for the per-client part is good enough to run a lot of communication rounds, however, the delta we get is very disproportional to the per-example delta (see Table 4.6). Although this is not a very big issue (especially as the dataset and the clients grow much larger), we notice that the per-client privacy part acts as a bottleneck to the training procedure; We could run many more training rounds without breaking the $\frac{1}{N}$ rule for the per-example privacy part. However, the per-client privacy part breaks the rule much faster, and so this hinders the training procedure to some extent. That being said, it would make sense to look into ways to achieve better system-wide privacy bounds such that more training rounds can be realized as this can potentially increase the accuracy metric further.

In addition, looking into concepts such as local DP [50] and methods like amplification by shuffling, see [29, 3] would also make sense. For example, amplification by sampling operates on disjoint partitions of the data (such as in our case), and it is based on shuffling the data points before applying DP-algorithms to them. Therefore, it could potentially provide stronger privacy guarantees than the ones presented in this thesis.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Devansh Arpit, Stanislaw Jastrzkebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 233–242. JMLR. org, 2017.

[3] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. The privacy blanket of the shuffle model. *arXiv preprint arXiv:1903.02837*, 2019.

[4] Daniel C Barth-Jones. The "re-identification" of governor william weld's medical information. *SSRN Electronic Journal*, 2012.

[5] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Differentially private empirical risk minimization: Efficient algorithms and tight error bounds. *arXiv preprint arXiv:1405.7085*, 2014.

[6] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[7] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*, 2018.

[8] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[9] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *Asian Conference on Machine Learning*, pages 97–112, 2011.

[10] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[11] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[13] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 267–284, 2019.

[14] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109, 2011.

[15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[18] Christos Dimitrakakis, Blaine Nelson, Aikaterini Mitrokotsa, and Benjamin IP Rubinstein. Robust and private bayesian inference. In *International Conference on Algorithmic Learning Theory*, pages 291–305. Springer, 2014.

[19] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[21] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[22] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[24] 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.

[25] Cynthia Dwork and Guy N Rothblum. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.

[26] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2010.

[27] Cynthia Dwork, Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Analyze gauss: optimal bounds for privacy-preserving principal component analysis. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 11–20. ACM, 2014.

[28] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016.

[29] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2468–2479. SIAM, 2019.

[30] Albrecht Fehske, Gerhard Fettweis, Jens Malmodin, and Gergely Biczok. The global footprint of mobile communications: The ecological and economic perspective. *IEEE Communications Magazine*, 49(8):55–62, 2011.

[31] 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. ACM, 2015.

[32] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[33] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[34] Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[36] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[37] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691*, 2017.

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

[39] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies*, 2019(1):133–152, 2019.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[41] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. 2018.

[42] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618. ACM, 2017.

[43] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[44] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[45] IBM. Accessed: 2019-02-25. https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/.

[46] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[47] 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.

[48] Prateek Jain and Abhradeep Guha Thakurta. (near) dimension independent risk bounds for differentially private learning. In *International Conference on Machine Learning*, pages 476–484, 2014.

[49] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1376–1385. JMLR. org, 2015.

[50] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[51] Daniel Kifer, Adam Smith, and Abhradeep Thakurta. Private convex empirical risk minimization and high-dimensional regression. In *Conference on Learning Theory*, pages 25–1, 2012.

[52] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

[54] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[55] 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.

[56] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[57] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[58] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[59] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[60] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[61] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[62] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, pages 6231–6239, 2017.

[63] Barbaro M. and Tom Zeller J. A face is exposed for aol searcher no. 4417749. *The New York Times*, Aug. 9 2006.

[64] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[65] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[66] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[67] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, volume 7, pages 94–103, 2007.

[68] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.

[69] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning, 2018.

[70] Marvin L Minsky and Seymour Papert. Perceptrons: an introduction to computational geometry. 1969.

[71] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017.

[72] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[73] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[74] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[75] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 ieee symposium on security and privacy*, pages 111–125. IEEE, 2008.

[76] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[77] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[78] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Ulfar Erlingsson. Scalable private learning with PATE. In *International Conference on Learning Representations*, 2018.

[79] Mijung Park, James Foulds, Kamalika Chaudhuri, and Max Welling. Variational bayes in private settings (vips). *arXiv preprint arXiv:1611.00340*, 2016.

[80] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[81] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. Knock knock, who's there? membership inference on aggregate location data. *arXiv preprint arXiv:1708.06145*, 2017.

[82] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard, 2019.

[83] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[84] Aaron Roth. New algorithms for preserving differential privacy. *Microsoft Research*, 2010.

[85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:9, 1986.

[86] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[87] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[88] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[89] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[90] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[91] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. *arXiv preprint arXiv:1710.10571*, 2017.

[92] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248. IEEE, 2013.

[93] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*, pages 3517–3529, 2017.

[94] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

[95] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[96] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[97] Latanya Sweeney, Akua Abu, and Julia Winn. Identifying participants in the personal genome project by name (a re-identification experiment). *arXiv preprint arXiv:1304.7605*, 2013.

[98] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[99] Om Thakkar, Galen Andrew, and H. Brendan McMahan. Differentially private learning with adaptive clipping, 2019.

[100] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660, 2014.

[101] Aleksei Triastcyn and Boi Faltings. Generating artificial data for private deep learning. *arXiv preprint arXiv:1803.03148*, 2018.

[102] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Towards demystifying membership inference attacks. *arXiv preprint arXiv:1807.09173*, 2018.

[103] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.

[104] Guido van Rossum. Python programming language. 01 2007.

[105] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[106] Yu-Xiang Wang, Stephen Fienberg, and Alex Smola. Privacy for free: Posterior sampling and stochastic gradient monte carlo. In *International Conference on Machine Learning*, pages 2493–2502, 2015.

[107] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[108] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[109] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[110] Xi Wu, Fengan Li, Arun Kumar, Kamalika Chaudhuri, Somesh Jha, and Jeffrey Naughton. Bolt-on differential privacy for scalable stochastic gradient descent-based analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1307–1322, New York, NY, USA, 2017. ACM.

[111] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282. IEEE, 2018.

[112] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[113] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[114] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[115] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[116] Ying Zhang, Mohammad Pezeshki, Philémon Brakel, Saizheng Zhang, Cesar Laurent Yoshua Bengio, and Aaron Courville. Towards end-to-end speech recognition with deep convolutional neural networks. *arXiv preprint arXiv:1701.02720*, 2017.

[117] S.K. Zhou, H. Greenspan, and D. Shen. *Deep Learning for Medical Image Analysis*. Elsevier Science, 2017.

Appendix A

Algorithms

Algorithm 4 Proposed Method for End-to-End Differential Privacy.

Input: Examples $\{x_1, ..., x_N\}$, loss function $\mathcal{L}(w) = \frac{1}{N}\sum_i \mathcal{L}(w, x_i)$. Parameters: learning rate η_t, noise scale σ, batch size B, gradient norm bound S, local epochs E, target epsilon per example ϵ_{ex}, target epsilon per client ϵ_{cl}, total clients K, fraction of participating clients per round C and $\mathcal{P}_k$ is the local data partition of client k with size $|\mathcal{P}_k|$.

Server Executes:
initialize w_0 randomly
$\sigma_{rec} \leftarrow \frac{\sigma}{\sqrt{\frac{|\mathcal{P}_k|}{B} \cdot E}}$
$m_t \leftarrow \max(C \cdot K, 1)$
for each round t =1,2,... **do**
  $Z_t \leftarrow$ (sample without replacement a set of m_t clients from K)
  for each client $k \in Z_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
  print privacy_spent(σ_{rec}, t, parameters)

ClientUpdate(k, w_t)**:** //Runs on Client k
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches b of size B)
for $t \in E$ **do**
  for batch $b_t \in \mathcal{B}$ **do**
    Compute gradient
    For each $i \in b_t$, compute $g_t(x_i) \leftarrow \nabla_{w_t}\mathcal{L}(w_t, x_i)$
    Clip Gradient
    $\bar{g}_t(x_i) \leftarrow g_t(x_i)/\max(1, \frac{||g_t(x_i)||_2}{S})$
    Add Noise
    $\tilde{g}_t \leftarrow \frac{1}{B}(\sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 S^2 \mathbf{I}))$
    Descent
    $w_{t+1} \leftarrow w_t - \eta_t \tilde{g}_t$
return w_{t+1}

Algorithm 5 Privacy Accounting.

Input: Moments accountant $\mathcal{M}$.

privacy_spent(σ_{rec}, t, parameters):

// Per Example Accounting:
Per-example Sampling rate (q_{ex}):
$q_{ex} \leftarrow \frac{B}{|\mathcal{P}_k|} \cdot C$
Per-example steps (st_{ex}):
$st_{ex} \leftarrow \frac{|\mathcal{P}_k|}{B} \cdot E \cdot t$
$\delta_{ex} = \mathcal{M}(\sigma, st_{ex}, q_{ex}, \epsilon_{ex})$

// Per Client Accounting:
Per-client Sampling rate (q_{cl}):
$q_{cl} \leftarrow C$
Per-client steps (st_{cl}):
$st_{cl} \leftarrow t$
$\delta_{cl} = \mathcal{M}(\sigma_{rec}, st_{cl}, q_{cl}, \epsilon_{cl})$

return δ_{ex}, δ_{cl}

Algorithm 6 Clipping Threshold Decay.

clipping_threshold ← high_value

for each round t =1,2,... **do**
  clipping_threshold ← max(clipping_threshold, low_value)
  train_clients(parameters, clipping_threshold)
  clipping_threshold ← clipping_threshold - clipping_threshold · decay_value

Algorithm 7 Trusted Curator. The K clients are indexed by k; B is the local minibatch size, C is the fraction of participating clients, E is the number of local epochs, $\mathcal{P}_k$ is the local data partition of client k, η is the learning rate & S corresponds to the clipping threshold.

Server Executes:

initialize w_0

for each round t =1,2,... **do**

$m_t \leftarrow \max(C_t \cdot K, 1)$

$Z_t \leftarrow$ (sample a set of m_t clients)

for each client $k \in Z_t$ **in parallel do**

$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

$w_{t+1} \leftarrow w_t + \frac{1}{m_t}(\sum_{k=1}^{K} w_{t+1}^k / \max(1, \frac{||w_{t+1}^k||_2}{S}) + \mathcal{N}(0, S^2 \cdot \sigma^2))$

ClientUpdate(k, w_t)**:** //Runs on Client k

$w \leftarrow w_t$

$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)

for each local epoch i from 1 to E **do**

for batch $b \in \mathcal{B}$ **do**

$w \leftarrow w - \eta \nabla \ell(w; b)$

$w_{t+1} = w - w_t$

return w_{t+1} to server

Algorithm 8 Naive Method for End-to-End Differential Privacy.

Input: Examples $\{x_1, ..., x_N\}$, loss function $\mathcal{L}(w) = \frac{1}{N}\sum_i \mathcal{L}(w, x_i)$. Parameters: learning rate η_t, noise scale (per example) σ_{ex}, noise scale (per client) σ_{cl}, batch size B, gradient norm bound S_{ex}, parameters norm bound S_{cl}, local epochs E, target epsilon per example ϵ_{ex}, target epsilon per client ϵ_{cl}, total clients K, fraction of participating clients per round C and $\mathcal{P}_k$ is the local data partition of client k with size $|\mathcal{P}_k|$.

Server Executes:
initialize w_0 randomly
$m_t \leftarrow \max(C \cdot K, 1)$
for each round t =1,2,... **do**
  $Z_t \leftarrow$ (sample without replacement a set of m_t clients from K)
  for each client $k \in Z_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow w_t + \frac{1}{m_t}(\sum_{k=1}^{K} w_{t+1}^k / \max(1, \frac{||w_{t+1}^k||_2}{S_{cl}}) + \mathcal{N}(0, S_{cl}^2 \cdot \sigma_{cl}^2))$
  print privacy_spent(σ_{cl}, t, parameters)

ClientUpdate(k, w_s)**:** //Runs on Client k
$w_t \leftarrow w_s$
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches b of size B)
for $t \in E$ **do**
  for batch $b_t \in \mathcal{B}$ **do**
    Compute gradient
    For each $i \in b_t$, compute $g_t(x_i) \leftarrow \nabla_{w_t}\mathcal{L}(w_t, x_i)$
    Clip Gradient
    $\bar{g}_t(x_i) \leftarrow g_t(x_i) / \max(1, \frac{||g_t(x_i)||_2}{S_{ex}})$
    Add Noise
    $\tilde{g}_t \leftarrow \frac{1}{B}(\sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma_{ex}^2 S_{ex}^2 \mathbf{I}))$
    Descent
    $w_{t+1} \leftarrow w_t - \eta_t \tilde{g}_t$
$w_{t+1} \leftarrow w_{t+1} - w_s$
return w_{t+1}

TRITA EECS-EX-2019:810