TECHNICAL UNIVERSITY OF CRETE

DEPARMENT OF ELECTRICAL & COMPUTER ENGINEERING

Differentially private data synthesis using Variational Autoencoders

Author
Margaritis Georgios

Thesis Committee
Prof. Minos Garofalakis (Supervisor)
Prof. Antonis Deligiannakis
Prof. Vasilis Samoladas

June 28, 2021

Abstract

Margaritis Georgios
Thesis Supervisor: Prof. Minos Garofalakis

Following major privacy breaches around the world, individuals and organizations are becoming increasingly reluctant in giving away their personal data. This heightened awareness for privacy is hindering the creation of rich, centralized datasets, and results in data owners keeping their datasets private. However, if different parties are unwilling to share their data with one another, then the models they will be able to build on their own will be of inferior quality, due to the lack of data.

In this thesis, we attempt to combine Variational Autoencoders, Federated Learning and Differential Privacy to solve this problem. These tools can enable a group of individuals or organizations to collaboratively create a rich synthetic dataset, without revealing their private data to one another, and without compromising their privacy. Then, they can all use the synthetic dataset to supplement their private datasets, they can use it to perform hyperparameter tuning on their models, or they can even release it publicly and share it with any other party. In any case, they will be mathematically assured that their privacy won't be adversely affected, no matter what they choose to do with the synthetic dataset, or who they choose to share it with. Those privacy guarantees, which stem from the mathematical properties of Differential Privacy, are crucial when dealing with owners of sensitive data such as hospitals and healthcare organizations. In such cases, the volume of data a single hospital has may be rather limited, potentially leading to very poor diagnostic models. Hence, a privacy-aware synthetic dataset created by multiple hospitals, could pave the way for much better diagnostic models, while preserving the privacy of hospitals and their patients.

Acknowledgements

After 5 amazing years at the Technical University of Crete, and before embarking on my research venture in the US, I want to express my deepest gratitude to the people that supported me throughout these years.

First and foremost, I would like to thank my supervisor, Prof. Minos Garofalakis for exposing me to very fascinating and niche research areas. Without his guidance and support, the completion of this thesis wouldn't be possible. I would also like to thank the members of his team at Athena Research Center, and particularly George Pikramenos, whose suggestions and insights on the topic were very valuable in completing my thesis.

I also owe a lot to Prof. Daphne Manoussaki for allowing me to serve as a teaching assistant and for helping tremendously with my doctoral applications: Her support was crucial when I had to make difficult choices and decisions. Additionally, I want to thank Prof. Liavas for supporting me and for exposing me to the interesting area of Optimization through his Convex Optimization courses. I also want to thank Prof. Aggelos Bletsas for his mentoring and advice, Prof. Vasilis Samoladas for his exciting courses on Distributed Systems and Computational Geometry, and Prof. George Karystinos for his great courses on Information Theory and Number Theory.

However, this journey wouldn't be possible without the help of my family and friends. I owe a lot to my parents who unconditionally supported me through every decision I made and gave me the opportunity to pursue all of my goals. I also want to express my gratitude for my close friends, who were there for me both in good and in bad times.

Contents

List of Abbreviations

URS	Uniform Random Sample
DP	Differential Privacy
VAE	Variational Autoencoder
GAN	Generative Adversarial Network
IID	Independent and Identically Distributed
FL	Federated Learning
ML	Machine Learning
PDF	Probability Density Function
MPC	Multi Party Communication
CDP	Central Differential Privacy
LDP	Local Differential Privacy

Glossary

Adversary: In cryptography, an adversary (rarely opponent, enemy) is a malicious entity whose aim is to prevent the users of the cryptosystem from achieving their goal.

Honest-but-curious adversary: The honest-but-curious (HBC) adversary is a legitimate participant in a communication protocol who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages [47].

Single-digit ε differential privacy: When we use the term "single-digit ε differential privacy" we refer to the case where we have (ε, δ)-DP guarantees with $\varepsilon < 10$. When $\varepsilon < 10$ and δ is small, the level of differential privacy is considered relatively good, and that's why many authors aim at enforcing DP with single-digit ε.

Chapter 1

Introduction

1.1 The call for privacy

In the era of Information and Ubiquitous computing, the importance of data in our lives cannot be understated. Every successful corporation has made computation and data-driven decision making an integral part of its business model. In fact, data has become a new form of currency. The vast majority of online platforms, such as Facebook, Google, YouTube e.t.c. utilize their users' data to generate revenue, while compensating for the platform's free service.

At the same time, we may not realize it, but by surfing through the internet, we give away vast amounts of our personal information. This wasn't a major concern, until the scandal of Cambridge Analytica proved that the implications of neglecting our privacy are far-reaching, and can even impact democratic institutions. Since then, there have been many global endeavors, such as GDPR, aimed at enforcing privacy constraints when processing people's data.

In light of all these concerns, individuals and organizations are becoming increasingly reluctant in giving away their personal data. This is creating a shift from traditional, centralized ML architectures to privacy-aware decentralized ML pipelines. In other words, there is an ongoing need for Machine Learning models that can be trained in a decentralized fashion, without requiring the data owners to give away their data. For instance, a clear manifestation of this necessity appears in the medical domain.

Due to the rapid advances in the field of healthcare analytics and personalized medicine, there has been an burgeoning demand for algorithms that utilize sensitive, biomedical data in order to extract various health-related insights. In particular, due to the scarcity and importance of biomedical datasets, there is an ongoing need for mechanisms that securely combine sensitive data from different sources (e.g. from different hospitals or healthcare organizations) in order to produce richer datasets and models that are extremely valuable not just for the data-holders themselves, but also for the entire research community.

Motivated by those attractive applications, we aspire to create a privacy-aware data-synthesis engine that can be trained on decentralized datasets. This system will be trained through the collaboration of multiple data-owners (e.g. individuals, organizations) who will actively participate in the training process, without sending their data to any other entity. Then, after the model is trained, the collaborating parties will know the exact privacy cost that their participation entailed. At the same time, the trained model can be used privately and securely by the training participants or any other entity as an artificial data generator that produces valuable, synthetic datasets, with quality similar to the training data.

In order to realize this goal, we will combine 3 different research areas: Federated

Learning, Differential Privacy and Generative Models. In particular, Generative Models will provide us with a trainable data-synthesis engine, Federated Learning will allow us to train this engine on distributed data, and Differential Privacy will enable us to mathematically quantify the privacy harm that this procedure incurs on data owners.

1.2 Federated Learning

Federated learning is a machine learning setting where multiple entities collaborate together to collectively train a machine learning model using their private datasets. One of the fundamental principles of Federated Learning is that the data owners never send their data to any party, but they instead actively participate in the training process by performing some local computations and sending their results to a centralized server. This property of Federated Learning makes it particularly privacy-friendly and attractive for many applications.

However, in order to be able to measure the privacy cost of someone's participation in a Federated Learning setting, we need to combine Federated Learning with Differential Privacy. This fusion allows the orchestrators of an FL setting to provide formal and mathematically rigorous privacy guarantees to all the parties that participate in the FL setting.

1.3 The promise of Differential Privacy

Differential privacy (or DP) is a state-of-the-art definition of privacy developed in 2006 by Cynthia Dwork. In Dwork's words, Differential Privacy describes a promise made by a data holder or curator to a data subject that the latter won't be affected, adversely or otherwise, by allowing their data to be used in any study or analysis, no matter what other studies, data sets, or information sources are available. In essence, Differential Privacy addresses the paradox of learning nothing about an individual while learning useful information about a population [25].

One important strength of Differential Privacy is that any differentially private algorithm, which is referred to as mechanism, remains differentially private no matter what an adversary knows and no matter what information is available elsewhere [23]. This means that an adversary cannot take the output of a differentially private algorithm and make it non differentially private.

The ultimate goal of Differential Privacy is to make sensitive data widely available for accurate analysis, without requiring strict protocols or agreements between data-holders and data-analysts. Unfortunately, however, there is an important trade-off between accuracy and privacy. In fact, according to the fundamental law of Information Recovery, requiring overly accurate answers to too many questions has an inevitable consequence on privacy. Nevertheless, the goal of a lot of algorithmic work in the area of Differential Privacy has been to mitigate this trade-off as much as possible by designing algorithms that maintain privacy without significantly compromising utility [25].

1.4 Generative Models

Generative models are unsupervised learning models which, when trained on a training set, have the property of generating synthetic data that are similar the training examples. This property makes generative models particularly useful in many real world scenarios, and especially in health-related applications where the supply of data is scarce.

Let's assume that we have a number of hospitals, where each hospital has a different part of a big database of clinical records of individuals. The subset of data that corresponds to every hospital is very small, and thus the hospital cannot train its diagnostic models solely by using its own data. Hence, if the hospitals managed to leverage each other's data, then they would be able to build much better diagnostic models. However, in most cases, there are strict policies which prohibit the hospitals from sharing their data with other parties.

Since data exchange is out of the question, we could instead leverage the power of Generative Models, Federated Learning and Differential Privacy to solve this problem. In particular, we can use a generative model to perform synthetic data generation, we can then use Federated Learning to train this model from decentralized datasets and we can combine all these with Differential Privacy so as to ensure that this process respects the privacy of the data owners.

The combination of Generative Models with Federated Learning and Differential Privacy, may enable organizations (such as hospitals) to utilize their limited data so as to jointly train a common Generative Model. Then, the participating organizations will be able to use the generative model so as to generate synthetic data and enrich their private datasets, without any fear for privacy. This way, the organizations will be able to significantly improve their models, by having access to rich synthetic datasets. By the same token, the synthetic data generated by the generative model can also be released to the public, without any privacy concerns. The reason for this is that since synthetic dataset and the generative model are differentially private, no adversary or malicious actor can make them less private. Hence, safely releasing the synthetic datasets the synthetic datasets to the public opens enormous possibilities for ways those datasets can be used. For instance, those datasets can be used by the research community to train models and extract various insights from rich but unforeseen data.

At the same time, one other important use-case for differentially private synthetic datasets is hyperparameter tuning. Let's assume that a group of organizations want to jointly train a classifier using their collective data, but without revealing their data to one another. This is a classic scenario where Federated Learning can be used to train the classifier. However, since the classifier is an ML model, it will require some level of hyperparameter tuning. Hence, in order to tune the hyperparameters of the classifier, we can set an initial value for the hyperparameters, train the classifier using FL on the real data, change the hyperparameters, train the classifier again using FL, and continue this iterative process until the model yields acceptable utility. However, the more times we perform FL on the real data, the greater the privacy loss from training. In particular, if the federated training of the classifier is differentially private, the more times we train the model on the real data, the worse the privacy guarantees get.

Hence, what we can instead do, is train a generative model on the distributed data using federated learning and differential privacy. Then, we can use the generative model to generate a differentially private synthetic dataset. This dataset can act as a summary/approximation of the real data and can then be used to tune the hyperparameters of the classifier. The most important benefit of this dataset, is that it can be used in as many experiments we want, without incurring additional privacy losses. This means that since the dataset is differentially private, it will retain the same level of privacy, no matter how many experiments we perform on this dataset. Consequently, we can train the classifier multiple times on this dataset and determine the best set of hyperparameters, without incurring any additional cost on privacy.

1.5 Thesis Organization

This work is organized in 5 chapters:

In **chapter 1**, we outline the necessity for privacy as an integral part of modern Machine Learning pipelines. We then explain the problem we are trying to address and its importance, especially in the medical domain.

In **chapter 2**, we explore the area of privacy-preserving analytics and its inherent challenges. Then, motivated by the shortcomings of traditional privacy approaches, we analyze the notion of Differential Privacy, while understanding its motivation, properties and mathematical primitives.

In **chapter 3**, we explore the domain of Federated Learning, its motivation and its interesting real-world applications. We then examine the privacy aspects of Federated Learning and its links with Differential Privacy.

In **chapter 4**, we examine the problem of differentially private, synthetic data generation from decentralized datasets. In particular, we propose a way to train Variational Autoencoders using Federated Learning and Differential Privacy. We then test our approach in various experimental settings and compare it against other approaches that attempt to solve the same problem as the one we are solving. In particular, our key contributions are the following:

- We manage to adapt Variational Autoencoders to a federated setting under Differential Privacy Guarantees. To our knowledge, this hasn't been achieved so far in a setting similar to ours. In fact, our proposed method for training federated Variational Autoencoders addresses the open problem of federating only the part of the Variational Autoencoder that is responsible for data synthesis (i.e. the decoder), while keeping the other part of the autoencoder (i.e the encoder) private [13].
- We reason about the privacy guarantees of our approach and present a concrete way to calculate the privacy budget.
- We test our approach on an image and a tabular dataset.
- We explore the relationship between privacy, utility, data distribution and number of network parameters.
- We compare our approach with other traditional approaches, such as PrivBayes and Federated DP GANs, and we demonstrate that Federated DP VAEs perform relatively well withing the scope of our experiments.

In **chapter 5**, we draw our conclusions, summarize our results and outline our contribution. We then talk about the drawbacks of our approach and suggest potential areas for future work.

Chapter 2

Privacy-preserving analytics

In this chapter, we are first going to explore the challenges of privacy-preserving analytics, while discussing the motivation behind privacy algorithms that rely on the notion of Differential Privacy. Then, we are going to analyze the definition of Differential Privacy, its properties and the various mathematical primitives, while also exploring how those primitives can be used in designing privacy-aware data-processing algorithms.

2.1 History of privacy-preserving analytics

2.1.1 Privacy Framework and the Sanitization Pipe Dream

In the classical (centralized) framework of privacy-preserving analytics, there is an agent (called curator) who has access to a sensitive database. This curator is trustworthy (i.e. has no intention of leaking sensitive data) and has sufficient computational power to protect the database from being hacked. Additionally, the database is assumed to be in a safe place and cannot be physically stolen. Hence, under those assumptions, the only way for somebody to have access to the database is through the curator. As a result, a malicious data analyst (adversary) can only infer information about a member of the database by relying on a statistical analysis of the data provided to him by the curator. Finally, to cover a wide range of senarios, we may also assume that the adversary has unlimited computational power and unlimited knowledge, meaning that apart from the information the adversary is trying to infer, he has access to any kind of additional databases [48].

In that setting, the ultimate goal of privacy-preserving analytics can be summarized in the "Sanitization Pipe Dream": A magical procedure is applied to a database that is full of useful and sensitive data. This procedure generates an "anonymized" or even a "synthetic" database that closely resembles the real database when it comes to data analysis, but at the same time maintains the privacy of everyone who was part of the original database [23]. This sanitization procedure is applied by the curator on the original database, and then the curator releases the sanitized database to the public, where the analysts can safely and accurately answer any query without infringing the privacy of the members of the original database (Figure (2.1.1)).

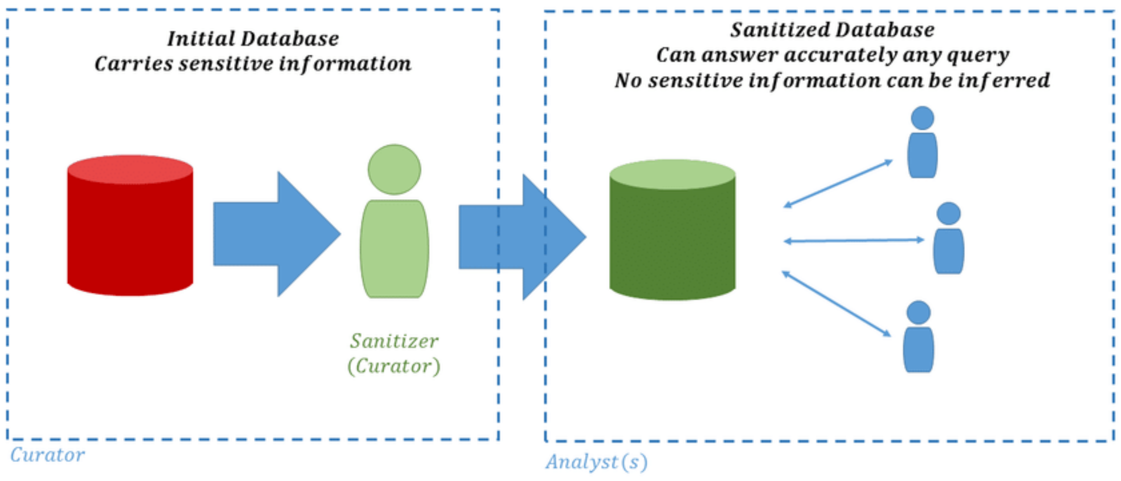

Initial Database
Carries sensitive information
Sanitizer
(Curator)
Curator
Sanitized Database
Can answer accurately any query
No sensitive information can be inferred
Analyst(s)

Figure 2.1.1: The Sanitization Pipe Dream

Unfortunately, however, it is a well known mathematical fact that requiring overly accurate answers to too many questions ultimately destroys privacy. In that sense, the sanitization pipe dream is clearly problematic: sanitization is supposed to give relatively accurate answers to all questions, but every system that does so, will inevitably compromise privacy [23].

2.1.2 Shortcomings of traditional privacy methods

Motivated by the sanitization pipe-dream and aspiring to create anonymized databases and datasets, there have been a lot of research attempts over the years in the area of privacy-preserving analytics. Unfortunately, however, most of those research attempts have failed to maintain acceptable levels of privacy. In part, their failure can be attributed to the prevalence of various adversarial attacks, which have managed to exploit supposedly "anonymized" databases and datasets:

1. *Linkage attack*: One traditional way that has been used to anonymize a set of records, was the removal of sensitive information (like name, phone number) , hoping that the remaining data attributes will not be sufficient to identify an individual from the population [25]. This notion of privacy, however, is problematic, as an adversary may attempt to identify individuals in the anonymized dataset by combining that data with non-anonymized public datasets. The 'linking' uses quasi-identifiers, such as zip , gender, date of birth, etc that are present in both datasets to establish identifying connections [3]. Those kinds of attacks are called "linkage attacks" and have been used to identify the medical records of the governor of Massachusetts, by combining anonymous health records with publicly available voter registration records. [25].

2. *Background Knowledge Attack*: In this attack, the adversary attempts to use an association between the sensitive attribute with quasi-identifier attributes (i.e. attributes available to the adversary) or background information about the target in order to eliminate possible values of the sensitive attribute. This attack is similar to Linkage Attacks, but instead of trying to identify whole rows, it tries to identify just the sensitive attribute of a particular row. For example, Machanavajjhala et.al. (2007) [41] showed that the information that heart attacks occur at a reduced rate in Japanese patients could be used to narrow the range of values for a sensitive attribute of a patient's disease [1].

3. *Differencing attack*: This kind of attack combines background information with statistics from overly accurate query answers in order to infer sensitive information about an individual. Although differencing attacks can become very complicated, in their simplest form, they require just 2 data points. For instance, if we have the background information that George's data have been included in a database of COVID-19 tests, then by asking "How many individuals have tested positive for COVID-19" and "How many individuals not named George have tested positive for COVID-19" we can probably ascertain whether George has tested positive for COVID-19 or not.

4. *Homogeneity attack*: In this type of attack, an adversary can leverage the case where all the values for a sensitive attribute within a group of records are identical. Hence, in such cases, the adversary can predict the value of the sensitive attribute in that group with 100% accuracy, despite the fact that the data may be anonymized.

Probably, one of the most popular anonymization methods has been k-anonymity, until it was proven to be an insecure way to anonymize datasets. The concept of k-anonymity was first proposed in 1998 by Latanya Sweeney et al. [49] and it has been used as a method to combat the risk of re-identification of anonymized data through linkage to other datasets [2] (i.e. protect against linkage attacks). In particular, a release of data about a population is said to have k-anonymity if the information for each person contained in the release cannot be distinguished from at least $k-1$ other individual whose information also appear in the release. K-anonymity can be enforced either by hiding the values of certain attributes (suppression) or by grouping different attribute values into the same category (generalization). The attributes available to an adversary are called quasi-identifiers and in a k-anonymous dataset, each quasi-identifier tuple occurs in at least k records.

However, as demonstrated in [41], this notion of anonymity is problematic, as it is susceptible to background knowledge attacks and homogeneity attacks [1]. This led to the proposition of the l-diversity privacy model [41], which is an extension of the k-anonymity and addresses some of the shortcomings of k-anonymity. In particular, although k-anonymity prevents the identification of a record against linkage attacks, it is not very effective in preventing the inference of some sensitive attributes of that record. In other words, a dataset anonymized with k-anonymity is safe from record linkage, but vulnerable to attribute linkage. Therefore, l-diversity was proposed as a way that not only maintains equivalence between at least k records, but also tries to enforce diversity in sensitive attributes, thus reducing the risk of homogeneity attacks. However, l-diversity doesn't come without its drawbacks. In fact, as demonstrated in [40] and [37], l-diversity and similar privacy models are susceptible to attacks that leverage the skewness in the distribution of the sensitive attributes (i.e. skewness attacks) and attacks that work when the sensitive attributes are distinct but semantically similar (Similarity Attacks)[27].

Besides k-anonymity and l-diversity, other anonymization methods have also been proposed, such as t-closeness, δ-presence, anatomy, (c, k)-safety e.t.c. However, most of those have proven to be vulnerable to some type of adversarial attacks. In fact, most traditional anonymization techniques offer a false sense of security, which has led to various breaches of privacy over the years. Many Organizations who released their datasets believing those datasets were anonymized, were subjected privacy attacks, resulting in leaks of sensitive information [2]. In fact, some of those leaks are so famous that have their-own short names: 'Governor Weld','AOL debacle','Netflix Prize' and 'GWAS allele frequencies' are just some of those.

In light of all these, an important question arises: Can we view the problem of privacy and anonymization from a different perspective?

2.1.3 Origins of Differential Privacy

The concept of Differential Privacy was inspired by the advances in cryptography of the 1970s-1980s. In "Pre-modern" cryptography, the prevailing paradigm in designing cryptosystems was the "propose-break-propose-again", where a cryptosystem was implemented, someone would break the cryptosystem, a new cryptosystem was then implemented and this procedure kept going again and again. [23]. In fact, this procedure was followed in designing most of the anonymization algorithms we mentioned above, as k-anonymity was proposed, then it was found that it is vulnerable to homogenity and background knowledge attacks, then l-diversity was proposed, then it was found to be vulnerable to skewness attacks, then t-closedness was proposed and the cycle kept repeating again and again.

Contrary to that approach, Goldwasser et al [28] stressed the importance of definitions when it comes to designing cryptosystems: First, the cryptographer defines the desired properties of the cryptosystem and bounds the computational capabilites of the adversary. Then, the cryptographer proposes the design of the cryptosystem and rigorously proves its security against all adversaries with the specified computational resources. This methodology converts a break of the implementation into a break of the definition of the cryptosystem; when an adversary manages to break the cryptosystem, then this means that there are weaknesses not in the implementation of the cryptosystem, but rather in the stated requirements of the cryptosystem (e.g. the adversary has more computational resources than assumed). This way, a break of the cryptosystem leads to the establishment of stronger and stronger definitions and requirements for the cryptosystem, inevitably leading to a path of progress [23].

Following this line of reasoning in the case of privacy, we may first want to explicitly state the definitions and requirements for privacy, and then try to design a system that provably meets those requirements. Hence, one way natural way to define privacy in data analytics could be to ensure that anything that can be learned about an individual through a privacy-preserving database can also be learned without access to the database. Unfortunately, this goal proposed by Dalenius [20] provably cannot be achieved, as we cannot expect to not learn anything new about any individual after a data analysis; if that was the case, then what would be the purpose of the analysis?

By relaxing Dalenius' definition, we may say that the goal of privacy preserving analytics is to ensure that the outcome of an analysis is the same, independently of whether an individual chooses to participate in the analysis or not. In other words, an individual should not be harmed more in the case he chooses to participate in the analysis compared to the case he chooses not to participate. In fact, this requirement is the backbone of Differential Privacy. Speaking informally, Differential Privacy states that the outcome of any analysis is essentially equally likely independent of whether any individual joins or refrains from joining the dataset [23].

2.2 Differential Privacy

Differential Privacy is a formal definition of privacy introduced by Dwork et al. [25]. Ever since it was proposed, Differential Privacy has become the state-of-the art definition of privacy with wide adoption not just from the academic world, but also from the industry as well. Differential Privacy provides protection against adversaries with unbounded computational resources and side information, while allowing the quantification of the privacy loss, even across multiple accesses to the same data [57].

2.2.1 Formal Definition

Let's assume that we have a database that can be modeled as a collection of rows, where each row corresponds to the data of a different individual. Differential privacy tries to ensure that the ability of an adversary to inflict harm or good to any subset of people stays the same no mater whether any individual is included or not in the dataset. For this reason, differential privacy examines all pairs of adjacent databases D and D' (i.e. databases that differ in exactly 1 record, such that a particular user's data is included in one database and not in the other) and tries to ensure that for all of those pairs of adjacent databases, the output of the differentially private algorithm (or to be more exact, the distribution of the output) stays almost the same.

Differential Privacy relies on randomized algorithms rather that on deterministic ones. Those algorithms are also called *mechanisms* and employ some part of randomness as part of their logic. Using that framework, the formal definition of Differential Privacy is the following:

Definition 2.1 *A randomized algorithm $\mathcal{M} : \mathcal{U} \rightarrow \mathcal{R}$ is $(\varepsilon, \delta)-$deferentially private if for all $\mathcal{S} \subseteq \mathcal{R}$ and all pairs of adjacent datasets D, D':*

$$P[\mathcal{M}(D) \in S] \leq e^{\varepsilon} P(\mathcal{M}(D') \in S) + \delta$$

where the probability space is over the coin flips of $\mathcal{M}$ [1].

In other words, if we consider as $\mathcal{R}$ the set of all the possible outputs that the randomized algorithm $\mathcal{M}$ can produce, then $(\varepsilon, 0)$-differential privacy indicates that any output $S \subseteq \mathcal{R}$ is almost equally likely to occur in any pair of adjacent database D and D'. When we say "almost equally likely", we mean that the probability of the the output S occuring in database D and the probability of S occuring in database D' can differ at most by a factor of e^{ε} (where for $\varepsilon \simeq 1$ then $e^{\varepsilon} \simeq 1 + \varepsilon$). This means that for a small ε, it is difficult to distinguish between database D and database D', as when the mechanism $\mathcal{M}$ operates in both databases, the outputs of the mechanism have (almost) the same probability distribution (Figure 2.2.1).

Adjacent Datasets

As we can see in the definition, Differential Privacy relies on the notion of *Adjacent Datasets*. So far, we have referred to adjacent datasets as datasets that differ in exactly one row (i.e. the row corresponding to a specific user). However, this is not always the case. In general, there are 2 different definitions for adjacent datasets, depending of the kind of differential privacy we want to enforce [43]:

Definition 2.2 *(Example-level Adjacent datasets): 2 datasets D and D' are called adjacent if D' can be formed by adding or removing a single training example from D. In other words, D and D' differ in exactly 1 record.*

Definition 2.3 *(User-level Adjacent datasets): Let D and D' be two datasets of training examples, where each example is associated with a user. Then, D and D' are adjacent if D' can be formed by adding or removing all of the examples associated with a single user from D.*

[1]Randomized algorithms can also be seen as deterministic algorithms which take 2 inputs: the data and a string of random bits. This random bit-string acts as the source of randomness and determines the probability space of the randomized algorithm. Hence, when we say that the probability space is over the coin-flips of mechanism $\mathcal{M}$, then we are implying that the probability space is dictated by all the possible values of the random bit string.

The first definition is mostly used when dealing with centralized datasets or databases, where we want to enforce example-level privacy. On the other hand, the second definition is mostly used in the federated setting where we want to enforce privacy guarantees that protect an entire client (e.g. hospital).

As a result, whenever we give differential privacy guarantees about an algorithm, we should be careful as to what kind of "adjacent datasets" we refer to. In essence, we have to distinguish between example-level privacy and user-level privacy, where the latter is much stronger than the former, as it includes much more records.

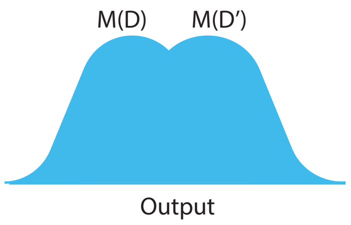
Parameter ε

The ε parameter of Differential Privacy is referred to as *privacy parameter* or *privacy budget* and acts as a metric of the privacy loss of a differentially private mechanism $\mathcal{M}$. We will discuss 2 different cases for epsilon in the case of $(\varepsilon, 0)$ Differential Privacy:

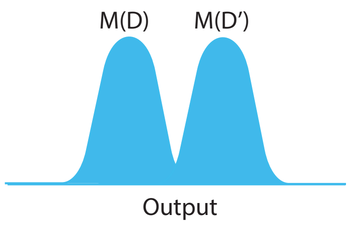
1. If ε is small, then this means that we have good DP guarantees. In essence, if we have a small value of ε (i.e. use a lot of noise) and then give the adversary the output of $\mathcal{M}$ on D and the output of $\mathcal{M}$ on D', then it will be difficult for the adversary to distinguish which output came from which database. The reason for this is that as we can see on Figure (2.2.1a), there is a significant overlap between the output distributions in the 2 cases, and thus if we draw a sample out of some of those distribution it may be difficult to distinguish which distribution this sample came from. In any case, it should be noted that "all small epsilons are alike"[25]. This means that DP mechanisms with small values of ε offer similar DP guarantees, and that if a mechanism fails to be $(\varepsilon, 0) - DP$ for a small value of ε, then this is not necessarily alarming, as the mechanism may be $(2\varepsilon, 0)$-DP for example

2. If ε is large, then this does not mean that we have bad privacy protection for every pair of adjacent databases D and D'. A large value of ε simply means that there exists at least one pair of adjacent databases and an output M, for which the ratio of the probabilities of observing M conditioned on the different databases is large. However, this output M may be very unlikely to occur (this is instead addressed by (ε, δ) differential privacy), the databases D and D' may be unlikely to appear in the real world, or the adversary may not have the right auxiliary information to recognize the revealing output M. Thus, this means that a large ε can cause anything from a small and meaningless privacy risk to a complete revelation of the entire database. In other words, a large epsilon is large in its own way [25].

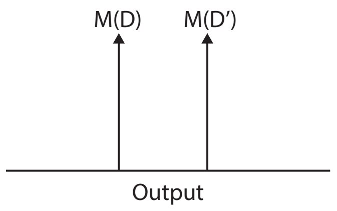
When it comes to the meaning of ε, we can also say that ε is an individual's limit of how much privacy they are ok with leaking. In fact, every time an individual participates in a differential privacy calculation, then this calculation reduces the individual's privacy budget by ε. Hence, if the individual participates in multiple such calculations, then the privacy loss accumulates. This means that although a single differentially private operation may not reveal anything about the individual, multiple such operations may be combined together to infer information about the individual. Consequently, when designing systems that rely on differential privacy, we have to be very careful so as to track the cumulative privacy loss of the differentially private mechanism and not just the loss in 1 DP operation.



(a) Low ε, high privacy, high noise (randomized)



(b) High ε, low privacy, low noise (randomized)



(c) $\varepsilon = \infty$, no privacy, no noise (deterministic)

Figure 2.2.1: PDFs of the output of $\mathcal{M}$ in D and D' for different ε

Parameter δ

The δ parameter of differential privacy is the probability of information accidentally being leaked. (ε, δ)-Differential Privacy is a relaxation of $(\varepsilon, 0)$-Differential Privacy which generally enables us to achieve lower values of ε if we allow δ to be positive instead of 0. Typically, we are interested in values of δ that are less than the inverse of any polynomial in the size of the database. In fact, values of δ that are in the order of $1/N$ (where N is the size of the database) are very dangerous, as they permit privacy by releasing the complete data of a small number of database participants [25].

At this point, it is worth mentioning that even for a very small δ, there is a distinction between $(\varepsilon, 0)$ differential privacy and (ε, δ) differential privacy. The main difference is that $(\varepsilon, 0)$-DP ensures that the output of a DP-mechanism is almost equally likely to be observed in every neighboring database. On the other hand, (ε, δ)-DP says that for every pair of adjacent databases D and D', it is extremely unlikely that an output $\mathcal{M}(D)$ is much more likely to occur in database D than on D'. However, it is possible to find an output $\xi \sim M(D)$ which is much more likely to occur on D than on D'.

In other words, if we define the quantity

$$\mathcal{L}^{(\xi)}_{\mathcal{M}(D)||\mathcal{M}(D')} = \ln\left(\frac{Pr[M(D) = \xi]}{Pr[M(D') = \xi]}\right) \tag{2.2.1}$$

and refer to it as the privacy loss incurred by observing ξ, then in $(\varepsilon, 0)$-DP this quantity is **always** bounded by ε, but in (ε, δ)-DP this quantity is bounded by ε with probability at least $1 - \delta$. So, there may be cases in (ε, δ)-DP where the privacy loss is not bounded by ε.

Utility-Privacy tradeoff

As we briefly mentioned, the goals of attaining privacy and utility are unfortunately not aligned. In fact, there is a significant trade-off between accuracy and privacy, which is one of the main reasons that hinder the universal adoption of differential privacy. Hence, although lower values of ε promise better privacy guarantees, they have a significant impact on utility. In fact, in order to attain low values of ε, we generally need to add a lot of noise to our DP mechanism, thus significantly reducing accuracy. However, a lot of research in the area of Differential Privacy is done to mitigate this trade-off as much as possible.

2.2.2 Properties

Despite the trade-off between accuracy and privacy, Differential Privacy offers several benefits when used as a privacy model, some of which are:

- **Worst case guarantees**
  Differential privacy offers a worst case rather than an average-case guarantee for privacy. This means that it protects privacy even in unusual scenarios and databases that are not likely to appear in real life, essentially protecting privacy even when dealing with outliers [23].

- **Quantification of privacy loss**
  One of the merits of differential privacy is that it gives us a mathematical framework to quantify privacy risk. This framework can be used to compare privacy loss inflicted by different privacy-preserving algorithms. Then, given 2 different algorithms that offer the same privacy guarantees we can ask, which has a better accuracy? In particular, the fact that an algorithm has certain privacy guarantees with a poor accuracy does not mean that all algorithms with the same privacy guarantees also have the same accuracy. In many cases, if we carefully design a privacy-preserving algorithm we can achieve better accuracy for the same level of privacy [23].

- **Building complex algorithms from primitives**: One important property stemming from the quantitative and cumulative nature of differential privacy, is that we can analyze the privacy loss of complex differential private algorithms by examining the privacy guarantees of their building blocks. By the same token, we can synthesize complex DP algorithms by combining simpler algorithms and primitives together [23].

2.2.3 Enforcing Differential Privacy

Let's assume that we have a function $f : \mathcal{U} \rightarrow \mathbb{R}^N$ that takes a dataset D and outputs a vector $f(D) \in \mathbb{R}^N$. Then, the ℓ_p sensitivity of f is defined as:

Definition 2.4 *(ℓ_p-sensitivity). The ℓ_p−sensitivity of a function $f : \mathcal{U} \rightarrow \mathbb{R}^N$ is:*

$$\Delta_p f = \max_{\mathrm{adj(D,D')}} \|f(D) - f(D')\|_p$$

Since f describes a deterministic algorithmic processing of D (where the output is always the same given the same input D), then by definition this process is not differentially private. Hence, to make this process differentially private, we have to apply a randomized mechanism on the output of f, essentially introducing some amount of noise.

Although there is a wide range of randomized mechanisms we can use, we will focus on the Laplacian and the Gaussian mechanisms, as they are the most widely used in the problem setting we are interested in. Hence, before we proceed, we will first describe the probability distributions of the Gaussian and the Laplacian distribution, as shown in definitions (2.5) and (2.6):

Definition 2.5 *(Gaussian Distribution) A random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ that follows the Gaussian distribution has probability density function:*

$$f_X(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \quad x \in \mathbb{R}$$

where $E[X] = \mu$ and $\mathrm{var}(X) = \sigma^2$.

Definition 2.6 *(Laplacian Distribution) A random variable $X \sim \mathrm{Laplace}(\mu, b)$ that follows the Laplacian distribution has probability density function:*

$$f_X(x|\mu,\sigma) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}, \quad x \in \mathbb{R}$$

where $E[X] = \mu$ and $\text{var}(X) = 2b^2$.

Given those probability distributions, we can add carefully calibrated noise to the output of the deterministic function f in order to make f differentially private. In particular, by introducing Laplacian noise to the output of f, we can achieve differential privacy through the Laplacian Mechanism, whereas by adding Gaussian noise, we can achieve differential privacy through the Gaussian Mechanism. Those 2 mechanisms are defined below:

Definition 2.7 *(The Gaussian Mechanism). Given any function $f : D \longrightarrow \mathbb{R}^k$, the gaussian mechanism is defined as:*

$$M_G(x, f(\cdot)) = f(x) + (Y_1, \dots, Y_k)$$

where Y_i are i.i.d. random variables drawn from $\mathcal{N}(0, \sigma^2)$

Definition 2.8 *(The Laplace Mechanism). Given any function $f : D \longrightarrow \mathbb{R}^k$, the laplacian mechanism is defined as [25]:*

$$M_L(x, f(\cdot), \varepsilon) = f(x) + (Y_1, \dots, Y_k)$$

where Y_i are i.i.d. random variables drawn from $\text{Laplace}(\Delta_1 f / \varepsilon)$

Since we have defined the Laplace and the Gaussian mechanisms, we can now reason about the DP guarantees that each one of those mechanisms achieve:

Theorem 2.1 *The Laplace Mechanism maintains $(\varepsilon, 0)$-differential privacy.*

Theorem 2.2 *Let $\varepsilon \in (0, 1)$ be arbitrary. For $c^2 > 2\ln(1.25/\delta)$, the Gaussian Mechanism with parameter $\sigma \geq c\Delta_2 f / \varepsilon$ is (ε, δ)-differentially private.*

Apart from Dwork's [25] Gaussian mechanism we mentioned above, there is also the Optimal Gaussian Mechanism described in [56] which achieves (ε, δ)-DP by introducing the least amount of Gaussian noise. The Optimal Gaussian Mechanism is described below:

Theorem 2.3 *(Optimal Gaussian mechanism (Opt-GM) for (ε, δ)-DP): The optimal Gaussian mechanism for (ε, δ)-differential privacy adds Gaussian noise with standard deviation σ to each dimension of a query with l_2-sensitivity Δ, for σ given by*

$$\sigma = \frac{\left(\xi + \sqrt{\xi^2 + \varepsilon}\right) \cdot \Delta}{\varepsilon\sqrt{2}} \tag{2.2.2}$$

where ξ is the solution of the equation

$$erfc(\xi) - e^{\varepsilon} erfc\left(\sqrt{\xi^2 + \varepsilon}\right) = 2\delta \tag{2.2.3}$$

and $erfc(\cdot)$ is the complementary error function.

As a result, both the Gaussian and the Laplacian mechanisms introduce some level of noise into the output of f to make it differentially private. The amount of noise that should be added depends on ℓ_1 and ℓ_2 sensitivity of f and also on the level of privacy we want to achieve; generally, for the same sensitivity, more noise leads to better privacy.

2.2.4 Closure under Post-processing

Probably, one of the most important properties of Differential Privacy is the closure under post-processing as described in [25]:

Theorem 2.4 *Let $\mathcal{M} : \mathcal{U} \longrightarrow \mathcal{R}$ be a randomized algorithm that is $(\varepsilon, \delta)-$deferentially private. Let $f : R \longrightarrow R'$ be an arbitrary randomized mapping. Then $f \circ \mathcal{M}$ is also (ε, δ) deferentially private.*

This theorem indicates that differential private databases are immune to post-processing. This means that if an adversary is given the output of a differentially private mechanism, then there is no way for him to make this output "less" differentially private, no matter what information or other resources he has at his disposal. This property is very important in our application as we want to train a differentially private generative model and then realese this model as an artificial data generator. Hence, if we manage to train our model with meaningful DP guarantees, then we can release it publicly, and no adversary will be able to make it less differentially private.

2.2.5 Composition

Another important set of primitives has to do with the composition of differentially private mechanisms. The use of composition theorems in differential privacy is motivated by several factors [24]:

1. Composition can be used to design and analyze complex DP mechanisms from simpler ones.
2. Composition can account for the repeated use of the same DP mechanism on the same database, in which case we want to make sure that the DP guarantees we get don't degrade much.
3. Composition can model the combination of different privacy mechanisms. For instance, if Bob's data were to be used by many DP mechanisms over time, we would want to to reason about the overall DP spending that those releases have had.

Given all these, we will focus on k-fold adaptive composition, where the data of a specific user (e.g. Bob) are used in k different differentially private data releases and we want to quantify the overall privacy budget spent. For this reason, we may use the weak composition theorem [24]:

Theorem 2.5 *The family of (ε, δ)-differentially private mechanisms satisfies $(k\varepsilon, k\delta)$-differential privacy under k-fold adaptive composition.*

In other words, if Bob's data were included in k data releases over time, then the theorem indicates that we need to have ε and δ smaller than $1/k$ in order to have meaningful DP guarantees over Bob's data. In fact, requiring δ to be that small is not a problem, as δ is usually negligible, but requiring ε to be that small has a significant impact on utility. However, if we are able to tolerate a negligible loss in δ, we may use the strong composition theorem to obtain better ε guarantees [24] :

Theorem 2.6 *For every $\varepsilon > 0, \delta, \delta' > 0$, and $k \in \mathbb{N}$,the class of (ε, δ)-differentially private mechanisms is $(\varepsilon', k\delta + \delta')$-differentially private under k-fold adaptive composition for*

$$\varepsilon' = \sqrt{2k \ln(1/\delta')}\varepsilon + k \cdot \varepsilon(e^{\varepsilon} - 1)$$

This theorem roughly gives $(O(\sqrt{k}\varepsilon), O(k\delta))$-DP guarantees for k compositions of the original DP mechanism [57].

2.2.6 Subsampling

Another important property of Differential Privacy that we will use throughout our thesis is the closure under the subsampling operation. In particular, we want to reason about what happens when an (ε, δ)- differentially private mechanism is applied on a uniform random sample (URS) of the original data.

First of all, we are going to define the subsampling procedure which creates a URS of a number of data points [57]:

Definition 2.9 *Given a dataset X of N points, the procedure "subsample" selects a random sample from the uniform distribution over all subsets of X of size m. The ratio $q := m/N$ is defined as the sampling parameter of the subsample procedure.*

Given this definition, we can now reason about the privacy guarantees of the subsampled mechanism [57]:

Lemma 2.1 *(Privacy Amplification under Subsampling) If $\mathcal{M}$ is $(\varepsilon, \delta) - DP$, then $\mathcal{M}'$ that applies $\mathcal{M} \circ$ subsample obeys (ε', δ')–DP with $\varepsilon' = \log(1 + \gamma(e^{\varepsilon} - 1))$*

This lemma describes a privacy amplification under subsampling and roughly states that subsampling with probability $q < 1$ amplifies an (ε, δ)-DP algorithm to an $(O(q\varepsilon), q\delta)$-DP algorithm for a sufficiently small choice of ε [57].

2.2.7 Moments accountant

In most cases, when DP is used in a deep learning setting, we can combine the privacy amplification lemma (2.1) with the strong composition theorem (2.6) in order to obtain DP guarantees for multiple training rounds of our model. However, as suggested in [8], this approach does not take into account the mechanics of the underlying noise distribution that is used during training, effectively providing weaker guarantees than what one could achieve.

Hence, Abadi et. al [8] proposed the use of a special data structure that is called Moments Accountant. This data structure keeps track of the cumulative privacy loss under multiple runs of a subsampled gaussian DP mechanism and is capable of providing tighter privacy guarantees than the strong composition theorem (2.6). In particular, the Moments Accountant keeps track of a bound on the moments of the random variable that determines the privacy loss. By doing that, the moments accountant offers $(O(q\varepsilon\sqrt{T}), \delta)$-DP guarantees over T runs of a (ε, δ) gaussian mechanism that samples the original data with sampling probability q at each step [8].

The Moments Accountant has been extensively used to keep track of privacy loss during Deep Learning and Federated Learning. However, improvements to the original moments accountant has also been suggested, such as the one proposed by Yu-Xiang Wang et. al. [57] which utilizes Renyi differential privacy to provide tight privacy guarantees over repeated applications of a subsampled Gaussian mechanism.

Chapter 3

Federated Learning

3.1 Introduction

Federated Learning is a machine learning scenario where multiple clients collaboratively train a machine learning model under the orchestration of a central server-coordinator. In Federated Learning, each client actively participates in the training process, without sharing its private data with any other entity. This property of Federated Learning guarantees that the data is kept decentralized and becomes crucial in mitigating privacy risks associated with training, especially if we compare Federated Learning to centralized ML pipelines. Additionally, due to the burgeoning IoT and distributed applications, there has been a lot of theoretical and practical research on Federated Learning, which has made FL a particularly niche research area over the past years [33].

The term federated learning was first proposed in 2016 by McMahan et al. [42]: "*We term our approach Federated Learning, since the learning task is soled by a loose federation of participating devices (which we refer to as clients) which are coordinated by a central server*". However, given that the original definition of FL focuses primarily on mobile devices and edge computing, there was a need to extend this definition to other settings as well, such as the setting where a relatively small number of more reliable clients (e.g. hospitals-organizations) collaborate to jointly train an ML model. According to the terminology introduced by Kairouz et al. [33], the FL scenario where mobile devices are involved is called cross-device FL, whereas the FL setting where organizations or larger entities are involved is called cross-silo FL. Hence, by trying to generalize the original definition of FL to include both of those settings, Kairouz et al. [33] proposed general definition:

Definition 3.1 *Federated learning is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective [33].*

Federated learning has countless applications and it has started to gain a lot of ground and scale, especially in the last decade. For instance, cross-device FL is extensively used by Google in the Gboard mobile keyboard and in Android messages. On the other hand, cross-silo applications of FL include finance risk predictions, pharmaceutical discovery, electronic health records mining, medical data segmentation and smart manufacturing. Nevertheless, Federated Learning poses many inherent challenges, such as the prevalence of unbalanced, non-IID datasets and the need to coordinate a large number of unreliable devices with limited bandwidth, particularly in cross-device FL settings[33].

3.1.1 Cross-device Federated Learning

In this variant of FL, our goal is to train a model using a very large number of mobile or IoT devices. Each client has its own training examples which never leave the client and no client can read the data of other clients. The whole process is orchestrated by a centralized server which never sees raw data [33].

Cross-device FL has its own sets of challenges. First of all, it relies on a vast number of clients (up to 10^{10}) which are not always available. The clients also tend to use slow communication networks (e.g. Wi-Fi) and are highly unreliable, as 5% or more of the clients participating to any given round are expected to fail due low levels of battery, weak networks or idleness [33].

3.1.2 Cross-Silo Federated Learning

The cross-silo setting refers to the case where a number of companies or organizations attempt to train a model on their collective data, but cannot do so by sharing their data directly. This restriction may be imposed, by legal constraints, by the need to maintain confidentiality, or even by the inability of a company to centralize its data from different geographical locations. Cross-Silo Federated settings typically involve the participation of $2-100$ clients. However, those clients are inherently much more reliable and trustworthy than the clients in cross-device FL [33].

In Cross-silo FL, the data partitioning among clients varies. In particular, there 2 types of partitioning: Partitioning by examples and partitioning by features. In the first case, every client's dataset has the same features (i.e. same columns) but different samples. In the second case, different clients may have different features for the same example. This, for instance, happens when we have different businesses with an overlapping set of customers, but where each one has different information on a particular customer [33].

Those 2 partitioning scenarios require very different training architectures and algorithms. For instance, in the case of partitioning by features, a centralized server may be not be involved at all in the training process. Instead, clients may exchange intermediate results with one another, while using techniques such as Secure Multi-party Computation or Homomorphic Encryption to limit the amount of information one client knows about the other through training [33].

However, in our work, we will solely address the case of partitioning by examples. This setting is used in cases where a company cannot centralize its data due to various constraints, or when organizations with similar goals want to collectively train and improve their models. For example, different banks may want to collaboratively train classifiers and models for fraud detections, hospitals can collectively build better diagnostic models etc [33].

3.2 Federated Training

In a usual federated learning scenario, the training process is orchestrated by a centralized server which repeats the following steps until the training is stopped [33]:

1. **Client selection**: The server randomly samples a number of clients from a list of potential clients. The selected clients, especially in cross-device FL, have to meet certain requirements imposed by the engineers who design the FL application. Such requirements might for instance be the availability of a client, the existence of a wi-fi connection, e.t.c.

2. **Broadcast**: The chosen clients download the current model weights and the training specification from the server.

3. **Client Computation**: Every chosen client computes a local weight update by performing a training step on its own data. In most cases, when we are dealing with neural network models, the clients use back-propagation in conjunction with Stochastic Gradient Descent to calculate the new weights.

4. **Aggregation:** The server collects and aggregates the local updates of the selected clients. In the case of SGD, the server may perform a federated averaging of the weights of the clients. Also, the step of aggregation is particularly important, as it is the stage where various techniques can be combined with FL, such as secure aggregation for increased privacy and noise addition in conjunction with gradient clipping for differential privacy.

5. **Model update:** The server updates the central model based on the aggregates of the updates of the participating clients.

Although the algorithmic process we described is very general and may be applied to arbitrary ML models, for the purpose of this thesis, we will solely focus on the training of Neural Networks using Federated Learning.

Neural Networks are a set of extremely powerful models which try to learn patterns from data, with a structure that imitates that of the human brain. Neural networks have been the catalysts of the AI revolution, with wide-range applications, in data classification, computer vision, image generation, natural language understanding and many other areas.

Generally speaking, a Neural Network can be viewed as a parametric function f_θ that takes an input $\mathbf{x} \in \mathbb{R}^n$ and produces an output $\mathbf{y} \in \mathbb{R}^m$ after a series of computations. The goal of training a neural network is to approximate an unknown function g based on the training data we have and the problem we are trying to solve. For instance, if we want to tell whether an image $\mathbf{x}$ depicts a cat or a a dog, then the function g we want to approximate is an unknown function that takes as input an image and decides if the image represents a cat or a dog. Then, since we don't know this function g, we want to approximate it with a neural network through training examples. In order to train a neural network, we need a cost function $l(\boldsymbol{\theta}, \mathbf{x})$ which measures how well our model performs. Then, we evaluate this function for every sample (or batch of samples) of the training set, and we use gradient based optimization algorithms (e.g. SGD) in conjunction with backpropagation in order to update the parameters $\boldsymbol{\theta}$ of the neural network.

The process we described above, briefly describes how a neural network is trained under a centralized setting (a thorough description would be out of the scope of this thesis). If we now wanted to train a Neural Network under a federated setting, he most common way to do that would be to use the Federated Averaging algorithm or FedAvg [42]. This algorithm is an adaptation of local-update or parallel SGD where each client executes a number of SGD update steps locally and then the local updates are averaged in order for the server to calculate the new weights for the global model. This procedure is shown in detail in Algorithm (1).

Algorithm 1: Federated Averaging

```
parameters: Total number of clients $N$, clients per round $M$, total communication
  rounds $T$, local steps per round $K$,number of samples in k-th client $nk$

Server executes:
Initialize model $\Theta^0$

foreach round $t \in \{1, 2, \ldots, T\}$ do
    $C^t \leftarrow$ (sample of M distinct users)
    foreach client $k \in C^t$ do
        $\Theta_k^{t+1} \leftarrow \text{ClientUpdate}(k, \Theta^t)$
    end
    $\Theta^{t+1} = \sum_{k \in C^t} \frac{n_k}{M} \Theta_k^{t+1}$
end

Function ClientTrain($k, \Theta^0$):
    parameters: Batch size $B \in \mathbb{N}$, learning rate $\eta \in \mathbb{R}^+$, client id $k$, loss function
      $l(\Theta; x)$, server weights $\Theta^0$, local epoch $E$

    foreach local epoch i from 1 to E do
        $\Theta_i \leftarrow \Theta_i^0$
        $\mathcal{B} \leftarrow$ (split data of the $k-$th client into batches of size $B$)
        foreach $b \in \mathcal{B}_i$ do
            $\Theta \leftarrow \Theta - \eta \nabla l(\Theta; b)$
        end
    end
    return $\Theta$;

End Function
```

3.3 Non-IID data in Federated Learning

One of the challenges of FL is that in most cases, the federated datasets are non-IID (i.e. they are not independent and identically distributed). In particular, when we refer to non-IID data in an FL setting, we generally mean that there are differences between data distribution $\mathcal{D}_i$ of client i versus data distribution $\mathcal{D}_j$ of client j. In most cases, this property stems from the fact that each client usually corresponds to a different user, a different geographic location or a different time window, which influences the data distributions different clients may have.

Generally, there are a lot of ways that client data can deviate from being IID. This may happen due to differences in feature distributions, label distributions, or differences in which features or labels are available in each client [33].

In real world FL scenarios, a combination of all those may appear and depending of type of "non-IIDness", different strategies may need to be employed in order to improve the results of FL [33].

3.4 Privacy in Federated Learning

Federated learning is a complex machine learning scenario that involves many different actors with radically different objectives. Ideally, in such scenarios, we would want every actor to have access to only the bare minimum amount of information that they need in order to fulfill their role. In addition to that, an ideal system would also enable its users to clearly and accurately know what kind of information might be revealed to others by participating in the system [33].

Designing a system that meets all of those requirements is a really challenging task on its own. However, FL has a range of desirable properties that favor the establishment of a useful and privacy-aware ML system. In particular, one of the inherent advantages of FL is that it provides a layer of privacy by not allowing the raw user data leave the user's device. Instead, only model updates are sent to the central server (e.g. gradients) which are strictly related to the ML task and typically convey significantly less information than the raw data. Also, the individual model updates only need to be stored to the server temporarily and not permanently, thus enhancing privacy even more [33].

Although these characteristics of FL do protect privacy to some extent, they don't offer rigorous privacy protection guarantees. In fact, there are instances where by knowing the model and the gradient updates from a client, the server can reconstruct training examples of that particular client, essentially inferring some portion of the raw data. Hence, in order to alleviate this problem, many techniques has been proposed, but we will mostly focus on Differential Privacy and Multi-Party Communication [33].

3.4.1 Actors & Threat models

As we discussed above, an FL scenario involves the interaction of different stakeholders with disparate goals. Hence, in order to characterize an FL system with respect to privacy, we normally have to define the threat model for the different entities that are involved and then offer privacy guarantees for each of those entities. For instance, we should differentiate the view of the server administrator from the view of a data analyst that uses the learned model, as even if we build a system that offers strong privacy guarantees against a malicious data analyst, then the same system may offer no guarantees against a malicious server. Hence, in most cases, in order to offer privacy guarantees to all the different actors that interact with an FL system, we may have to combine different privacy and security techniques together [33]. However, before we proceed, we will first mention some of threat models an FL system may have to fortify against:

- *Client access*: Someone who has access to a client device is assumed to be able to inspect all messages received from the server, including snapshots of the model that the server sends to the client during training. At the same time, they may also be able to tamper with the training process by interfering with the local updates sent to the server. However, if they are honest-but-curious, they may just observe the message exchanges, without interfering with the training process.

- *Server access*: Someone who has access to the server device is assumed to be able to inspect all messages (e.g. gradient updates) sent to the server during all rounds of FL and can therefore tamper with the training process. However, if they are honest-but-curious, they may just observe the message exchanges, without interfering with the training process.

- *Intermediate Model Access*: A data analyst or data engineer may have access to multiple states of the model from training settings with potentially different hyperparameters.

- *Deployed Model Access*: In cross-device FL, the model resulting from training may be deployed to hundreds of millions devices. Hence, if a device is compromised, then a malicious actor (e.g. hacker) may gain black-box or white-box access to the model, depending on the level of access they have on the device.

In order to combat some of those problems, FL can be used in conjunction with Differential Privacy and secure aggregation protocols.

3.4.2 Secure aggregation & SMPC

As we previously discussed, in a Federated Learning Setting, the centralized server-coordinator acts as an aggregator of the local client updates. However, in many cases, the server cannot be fully trusted. Hence, what we ideally want would be for the server to compute a sum or average of the updates sent by the clients, but without having access to the individual updates each client sent. This way, even if the server is compromised, the malicious actor that has access to the server will only know the aggregates of the client updates and not the individual updates of any particular client. This process is referred to as *secure aggregation* and enables each client to submit a value (such as a vector or tensor) so that the server learns nothing but an aggregate of the clients' values.

There has been a lot of literature on ways of achieving secure aggregation, such as the use of Secure Multi-Party Computation protocols [33]. In particular, secure multi-party computation (SMPC) is a sub-field of cryptography that enables distinct parties to jointly compute a function using their private inputs. Only the output of that function is made public and the parties don't learn anything more than their own inputs except whatever they can learn from the output of the function [26]. This field was first introduced in 1980's by Yao and thanks to various theoretical and engineering breakthroughs, it has led to technologies that were adopted by the industry [33].

Generally, cryptographic approaches such as the ones used for Multi-Party Computation rely on operations done in finite fields (i.e. integers modulo a prime number p), which often makes it difficult to represent real numbers. Hence, when such cryptographic approaches are combined with ML, normalization and careful quantization is generally used in the training process so as to avoid arithmetic overflows and underflows [33].

For several years, it has been known that any function can be securely computed in the presence of malicious adversaries. However, while generic solutions do exist, their real-world performance hinders their adoption in most practical scenarios. Hence, a lot of research has been devoted to designing MPC protocols for custom ML applications, such as linear regression, logistic regression and neural network training and inference. This works are generally concerned with the cross-silo FL setting, where a small number of participants are involved. In fact, the adaptation of SMPC protocols to a cross-device FL setting with millions of participants is a daunting task, due to the high computation and communication overhead [33].

One notable work that employs SMPC to achieve secure aggregation within the context of Federated Learning was done by Bonawitz et al. [16]. This work proposes an SMPC protocol that can securely aggregate m-dimensional vectors from n users with a computational cost of $O(n^2 + mn)$ and a communication cost of $O(n + m)$ on the user-side. For instance, the authors indicate that a population of $n = 2^{14}$ users can securely aggregate 2^{24}-dimensional vectors using a 1.98 expansion in communication, indicating a relatively low communication overhead, even for vectors and client pools. In fact, this work has also been integrated into Tensorflow Federated, which facilitates the usage of secure aggregation in FL settings.

Another very important work in the area of Secure Multi-Party Computation is the SPDZ framework introduced by Damgard et al. [21]. This work proposes a general SMPC protocol that can be used to securely compute a function over any finite field $\mathbb{F}_p^k$. The protocol consists of 2 phases: a preprocessing phase and an online phase. The preprocessing phase is independent of the function to be computed and independent of the inputs of the function. In particular, the preprocessing phase has a complexity of $O(n^2/s)$ operations, where n is the number of clients and s is a parameter that increases with the security parameter of the cryptosystem. On the other hand, the online phase is much more efficient than the preprocessing phase, featuring a computation and communication complexity linear in n, whereas previous works featured a complexity quadratic in n. Additionally, several works have been devoted to applying the SPDZ framework to an ML setting, such as the one by Chen et al. [17], where the authors use the SPDZ framework to perform a linear and a logistic regression using mini-batch SGD.

3.4.3 Differential Privacy

As we mentioned in the previous chapter, Differential Privacy is the state-of-the art definition of privacy that enables limiting and quantifying the privacy risks associated with an individual's participation in data disclosures. In the context of FL, since we are interested in user-level privacy, we will always use the definition (2.3) of user-adjacent datasets to dictate the type of privacy we are aiming for. In particular, in an FL setting we want to provide privacy guarantees for whole clients, essentially protecting all the training examples associated with a particular client and not just 1 training example (which would be the case in example-level privacy).

Central Differential Privacy

Over the past years, there has been a lot of work in the intersection of FL with Differential Privacy. In a non-FL setting, in order to enforce DP, the common assumption is that the raw data is collected centrally by a trusted authority (i.e. database curator) who is responsible for applying the necessary noise perturbations in order to achieve privacy. By the same token, this centralized differential privacy model can also be applied to an FL setting.

In many cases, the assumption we make in an FL setting is that there is a trusted central server that orchestrates the training process. This server, apart from aggregating the model updates from clients, is also responsible for applying the necessary DP mechanisms in order to guarantee the users' privacy. At the same time, the server also ensures that only the privatized versions of the trained model are released to other parties (e.g. model engineers, data analysts) [33].

Generally, there have been a lot works that train deep learning models by combining central DP with FL, such as the training of federated DP RNN models in [43] and the training of Federated DP GANs in [13]. In those works and similar ones, the main techniques used to achieve differential privacy is a combination of clipping the weight updates performed by clients and introducing Gaussian noise during the federated aggregation phase of training. Then, the privacy guarantees can be measured by using the Moments Accountant data structure we mentioned in the previous chapter. For example, Algorithm (2) which is based on [13] and [43] shows how we can use federated averaging in conjunction with central DP in order to train a neural network in a federated setting.

Algorithm 2: Federated DP Averaging

```
parameters: Total number of clients $N$, clients per round $M$, total
  communication rounds $T$, local steps per round $K$,round participation fraction
  $q$, clipping parameters $S$, noise scale $z$

Server executes:
Initialize model $\Theta^0$, moments accountant $\mathcal{M}$

$\sigma \leftarrow \frac{zS}{qN}$
$\mathcal{M} \leftarrow InitializeMomentsAccountant$
foreach round $t \in \{1, 2, \ldots, T\}$ do
    $C^t \leftarrow$ (sample of qN distinct users)
    foreach client $k \in C^t$ do
        $\Delta_k^{t+1} \leftarrow$ ClientUpdate($k, \Theta^t$)
    end
    $\Theta^{t+1} = \Theta^t + \frac{1}{qN} \sum_{k \in C^t} \Delta_k^{t+1} + \mathcal{N}(0, I\sigma^2)$
    $\mathcal{M}$.compose_subsampled_mechanism($z, q$)
end
print $\mathcal{M}$.get_epsilon($\delta$)

Function ClientTrain($k, \Theta^0$):
    parameters: Batch size $B \in \mathbb{N}$, learning rate $\eta \in \mathbb{R}^+$, client id $k$, loss function
      $l(\Theta; x)$, server weights $\Theta^0$, local epoch $E$

    $\Theta \leftarrow \Theta^0$
    $\mathcal{B} \leftarrow$ (split data of the $k-$th client into batches of size $B$)
    foreach $b \in \mathcal{B}_i$ do
        $\Theta \leftarrow \Theta - \eta \nabla l(\Theta; b)$
    end
    $\Delta_k = \Theta - \Theta^0$;
    $\Delta_k = \Delta_k \cdot \min\left(1, \frac{S}{\|\Delta_k\|}\right)$ //Gradient Clipping
    return $\Delta_k$
End Function
```

Local Differential Privacy

Although Central Differential Privacy may sometimes be sufficient, in many real-world applications, we may want to curb the need for a trusted third party (i.e. server) in a differentially private FL setting, as the existence of such party may be a rather strong assumption to make. For this reason, there exists the model of *Local differential privacy* (LDP), where the clients apply differentially private transformation to their data prior to sharing the model updates with the server. LDP has been used in a federated setting to train a spam classifier by Snap and has also been used to gather statistics across large user groups by Google, Apple and Microsoft. Unfortunately, maintaining privacy under LDP requires the participation of a much larger number of clients than the centralized DP setting. At the same time, pure LDP makes it very difficult to train a model while also maintaining utility. For this reason, hybrid methods has also been proposed, such as Distributed Differential Privacy and Hybrid Differential Privacy [33].

Distributed Differential Privacy

Distributed Differential Privacy offers some of the utility benefits of Central DP but without the assumption of a trustworthy central server. In Distributed DP, the clients send a minimal and encoded version of their updates to a secure computation function, whose output is made available to the central server. However, by the time the server gains access to the output of the secure function, this output already satisfies differential privacy guarantees. This way, the server has access solely to a noisy aggregate of the client updates and not on the individual updates themselves. However, Distributed DP does require some rather strong assumptions, such as the availability of an SMPC mechanism to perform the secure computation [33].

One of the methods we can use to achieve Distributed DP is secure aggregation. As we mentioned above, secure aggregation ensures that the central server obtains only the aggregated result and not the individual client updates or any other intermediate parameter that may be used to reveal information about a client. Also, in that setting, each client can introduce a moderate amount of noise into their updates so that the aggregated results obeys certain DP guarantees. In fact, if the noise added by each client is calibrated carefully, then both privacy and utility will reach similar levels as in Central DP, but with the added benefit that we make no assumptions about the existence of a trusted server [33].

Comparison of privacy models

Based on the way Central and Local Differential Privacy work, and taking into consideration that for each one of those we can enforce user-level DP or record-level DP, we can identify the following privacy models [60]:

- **ULDP**: (user-level privacy protection with distributed pertubation). In ULDP, each user i selects its own privacy parameters ε_i and δ_i and uses a local DP mechanism Y_i , such that given 2 instances x_i and x'_i of user i's data (which are user neighboring), then for any possible subset of outputs $\mathcal{Y}_i$ of Y_i, it must hold that $P(Y_i(x_i) \in \mathcal{Y}_i) \leq P(Y_i(x'_i) \in \mathcal{Y}_i)e^{\varepsilon_i} + \delta_i$. This privacy paradigm is the one that is used to achieve user-level local differential privacy.

- **ULCP**: (user-level privacy protection with centralized pertubation). In ULDP, the aggregator chooses privacy parameters ε and δ and applies a randomized mechanism $\mathcal{M}$ such that for any user i, for any 2 instances of the user i's data (which are user-neighboring) and for any possible subset of outputs A of $\mathcal{M}$, it must hold that $P(\mathcal{M}(x_i) \in A) \leq P(\mathcal{M}(x'_i) \in A)e^{\varepsilon} + \delta$. This privacy paradigm ensures user-level central differential privacy and it is weaker than ULDP as it makes the assumption of a trusted server-aggregator.

- **RLDP**(record-level privacy protection with distributed perturbation): This scenario is the same as ULDP, but instead uses a different definition of adjacent datasets, where the datasets x_i and x'_i are said to be adjacent if they differ in exactly 1 record. The RLDP privacy paradigm enforces record-level local differential privacy. However, this type of privacy is weaker than ULDP, as it protects the individual rows of a user's data and not all the data at once, as ULDP does.

- **RLCP**(record-level privacy protection with centralized perturbation): This privacy paradigm is the same as ULCP, but uses the definition of record adjacent datasets instead of user-adjacent datasets. Hence, in RLCP, the datasets x_i and x'_i are said to be adjacent if they differ in exactly 1 record. The RLCP privacy paradigm enforces record-level central differential privacy. However, this type of privacy is the weakest of all, as it protects only individual rows of a client's data and not the client's whole data at once, and at the same time, it assumes the existence of a trusted server which performs the centralized pertubation.

Following [60], we performed a comparison of the privacy properties of ULDP, ULCP, RLDP and RLCP as shown in Table (3.4.1). The comparison takes into account 3 factors:

1. **Protection against Honest but Curious Aggregator**. In this comparison dimension, we want to examine if the privacy model assumes that the clients fully trust the server to ensure Differential Privacy. In particular, we want to explore whether the privacy model offers protection against an honest-but-curious aggregator, who tries to learn information about the clients without interfering with the training protocol[1].

2. **Protection against attacks after model publishing**: This comparison dimension examines whether the different privacy paradigms protect privacy when an adversary has black-box or white-box access to the trained model. Generally, one of the main benefits of Differential Privacy , is the fact that the output of a DP mechanism cannot be made less Differentially Private (Theorem 2.4). Hence, by training a model under DP and releasing it publicly, no adversary can reduce the DP level of the model.

3. **Protections of a user's entire dataset**: This comparison dimension outlines the difference between user-level differential privacy and the weaker record-level differential privacy. In most FL scenarios, it makes more sense to enforce DP on a user-level rather than on a record-level, as user-level DP protects all the data of a particular user simultaneously, whereas record-level DP protects individual rows. Hence, in FL scenarios where a particular client has many data points associated with the same individual, it is much safer to use user-level DP. For instance, a hospital may have many data points about a specific patient, or a mobile device may have many training examples for its user, thus making the use of user-level DP paramount.

Privacy Paradigm	**Protects against Honest but Curious Aggregator**	**Protects against attacks after model publishing**	**Protects a user's entire dataset at once**
ULDP	✓	✓	✓
ULCP	✗	✓	✓
RLDP	✓	✓	✗
RLCP	✗	✓	✗

Table 3.4.1: Comparison of different FL privacy models

Throughout this work, we will only focus on ULDP and ULCP, as we want to enforce user-level DP and not the weaker record-level DP. Hence, from now on, whenever we refer to Central Differential Privacy, we will imply User-level Differential Privacy with Centralized Perturbation (ULCP) and whenever we refer to Local Differential Privacy we will imply User-level Differential Privacy with Distributed Perturbation (ULDP).

[1]We are not examining cases where aggregator interferes with the training protocol, as those cases are out of the scope of this thesis.

Chapter 4

Towards Federating Variational Autoencoders

In this chapter, we will first discuss the motivation behind federating generative models using differential privacy. Then we will focus on Variational Autoencoders as a class of generative models and we will discuss their widespread applications, their inner workings and interesting variations. At the same time, as the core subject of our thesis, we will analyze how we federated Variational Autoencoders using differential privacy and we will also present the experimental setting we used and the results that came out from our work.

4.1 Motivation

Over the past decade, generative models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), have become a state-of-the-art research area in Deep Learning. The reason for this is that they offer unique mechanisms for understanding and analyzing the rapidly expanding class of unlabeled datasets [45]. In particular, the main goal of generative models is to capture the probability distributions of a set of data points so as to be able to generate synthetic data similar to the training examples. Generative Models have been widely used in many areas, with applications in computer vision, speech recognition and generation, natural language processing, robotics and many others [45].

At the same time, although the era of information and ubiquitous computing has produced immense volumes of data, there are still domains where the supply of data is very limited. For example, in medical applications, when conducting a patient-level analysis, each patient is usually treated as a single sample in the training process of a model. Hence, since there are many different diseases and combinations of those diseases, the number of patients in the whole world is in most cases insufficient to produce acceptable volumes of data. As a result, medical datasets are in many cases unsuitable for directly training data-voracious models such as Neural Networks. Moreover, since medical datasets contain very sensitive information, having direct access to those data may be in many cases impossible due to serious privacy concerns [58].

Fortunately Differentially Private Federated Generative Models may be the answer to both of those problems. By learning the underlying data distributions from small training datasets, generative models have been proven to be very efficient ways of generating high-fidelity artificial data that can be used to enrich the original datasets. Hence, the combination of Generative Models with Federated Learning, may enable multiple organizations (such as hospitals) to utilize their limited data so as to jointly train a common Generative Model. Not only will this benefit the collaborating parties by providing them with ways fo enriching their original datasets, but it may also benefit the entire research community, by granting them access to an artificial generator of valuable but unforeseen data. Consequently, training Generative Models through Federated Learning may provide a promising solution for the data-scarcity problem; Although each organization may have limited amounts of data individually, the jointly trained Generative Model will learn a common, underlying distribution that will be sampled to generate new, synthetic data of quality similar to the real data.

The benefit of the federated setting we described is that the participating organizations won't need to send their data elsewhere in order for the training to happen. Instead, they will be actively involved in the whole process by training a model locally with their own data and then sending the gradient updates to the server. Unfortunately, this doesn't fully resolve the problem of privacy, but that's where Differential Privacy comes into play. The combination of Federated Learning with Differential Privacy can offer rigorous privacy guarantees to all participants by mathematically ensuring that everyone's participation in the model training changes very little the final result. This requirement which lies in the core of Differential Privacy is essential to ensure that no significant information about the data owners is leaked during training. Then, if the training process is done under DP guarantees, the model that will be generated after training will also follow the same DP guarantees. This way, the final generative model can be released to other parties or even publicly without any privacy risk. At the same time, the synthetic datasets generated from this process can be used to train a wide variety of models, can be analyzed to provide valuable data-related insights, or can be utilized to securely enrich the clients' private datasets.

At the same time, a Differentially Private synthetic data generator can also be very useful when tuning the hyperparameters of a federated model. If we want to train a classifier on federated data, then in order to determine the hyperparameters of this classifier, we generally need to run multiple training experiments, while studying the effect that the different hyperparameters have on the accuracy of the classifier. However, if those experiments are run directly on the real federated data, then the more experiments we perform, the less private our process becomes. For instance, if a single experiment of training the classifier on real data is (ε, δ)-differentially-private, then k runs of that experiment are $(k\varepsilon, k\delta)$-differentially-private. This means that there is a considerable increase in the privacy budget for each run of the experiment, which may limit the amount of experiments we can perform (e.g. data-owners may refuse to participate in further experiments if their privacy budget exceeds a certain threshold). On the other hand, if a synthetic DP dataset is used instead, this won't be an issue. If we first train a generative model on the real data using Differential Privacy and then generate a synthetic dataset, then this dataset can be used in as many experiments as we want, without incurring additional privacy losses, apart from the privacy loss occurred during training the generative model. This means that if the generative model is (ε, δ)-differentially-private, then the synthetic dataset will also be (ε, δ)-differentially-private (due to the post-processing lemma of differential privacy) and then, no matter how many experiments we run on the synthetic dataset, the process will remain (ε, δ)-differentially-private. Hence, we can run multiple experiments on the synthetic datasets without incurring additional privacy losses, and then, after we have determined the best hyperparameters for the classifier, we can train the classifier on the real data once.

In light of all these, federated generative models seem to have many advantages, and that's why we we chose to federate Variational Autoencoders for the purpose of this thesis. Variational Autoencoders are becoming increasingly popular among the research community both for their strong probabilistic foundation and their ability to produce meaningful latent representations of data [12]. VAEs have been used in many different and exciting applications, such as for brain aging analysis [59], anomaly detection [10], recommender systems [38] and even for molecular design [39]. At the same time, as seen in the work done by Lei Xu et. al [36], Variational Autoencoders demonstrate very solid results in centralized, tabular data synthesis, surpassing in most cases all other models, such as GANs and PrivBN. Hence, all those factors make Variatitonal Autoencoders very attractive candidates for performing Synthetic data generation from distributed datasets.

Last but not least, to our knowledge, Variational Autoencoders have not been used for differentially private data synthesis from distributed datasets before. This makes the combination of FL, DP and Variational Autoencoders a particularly interesting research topic with various open questions that are worthy of exploration. For instance, one such question is whether we can federate only one part of the Variational Autoencoder (i.e. the decoder), while keeping the other part of the VAE private (i.e the encoder). Such question arises from the fact that in order to perform synthetic data generation, we just need a trained decoder: the encoder is not used in data synthesis. Hence, if we allow the clients to keep their encoders private and share only their decoders, then we may achieve better DP guarantees and better quality of synthetic data than sharing both the encoders and the decoders. The reason for this is that the more information the clients keep private, the better privacy guarantees they can get, and at the same time, a private encoder for each client may adapt much better to the nuances of a client's own dataset than a global encoder that will be used by all clients.

4.2 Related Work

Our work combines three key components: generative models, federated learning and differential privacy. Most previous works in the area focus on at most 2 of those 3 components. For instance, there has been a lot of work on training generative models under Differential Privacy, but on centralized rather than on federated settings (e.g. Torkzadehmahani et al. [52], Xie et al. [58] and Takahashi et al. [51]). Also, there have been some works that focus on generative models for decentralized data problems but without any privacy guarantees. For instance, Hardy et al. [31] trained a GAN using decentralized data, albeit with no privacy protections.

When it comes to VAEs, there has been one work that combines VAEs with differential privacy using semi-distributed data [9]. However, this work uses a very different setting than ours, as it clusters a centralized dataset and then merges the clusters using VAEs, while offering record-level DP guarantees rather than the stronger user-level guarantees we are interested in.

Clearly, a previous work that is very close to ours is the one done by McMahan et al. [13], where the authors train a GAN on a federated setting, while ensuring user-level differential privacy. The federated setting the authors use is very similar to ours, but their goal when using a federated GAN is very different: They are trying to detect bugs in clients, whereas in our case, we are trying to create a DP data synthesizer that will generate data similar to the original data of the clients. Hence, in our case, we also have to synthesize the labels of the different data points, whereas in [13] this is not needed. Last but not least, in our case, we use a different way of measuring the quality of the generated data, which involves the Student Network (will be discussed in detail in a next section). However, the way the authors federated GANs inspired our work, and thus we will briefly describe their approach in section (4.2.1). Additionally, in next sections, we will also attempt to compare the Federated GANs proposed by the authors against our Federated VAEs in a data-synthesis setting.

4.2.1 Generative Adversarial Networks

GANs (Generative Adversarial Networks) are considered state-of-the-art generative models nowadays, with immense applications in generating image,text, sound and even videos (deepfakes). In fact, in a seminar of 2016, Yann LeCun, one of the fathers of AI, has called GANs "the coolest idea in deep learning in the last 20 years".

GANs were proposed by Ian Goodfellow et al. [29] and have their theoretical foundation on Game Theory. In particular, the fundamental principle of GANs is a two player minimax game between a neural network called Generator and a neural network called Discriminator. The generator attempts to fool the discriminator by generating realistic synthetic images, while the discriminator tries to distinguish real images from fake ones, as shown in Figure (4.2.1).

As the training progresses, the generator gradually becomes better and better at creating realistic images, while the discriminator becomes better at distinguishing them from real ones. In theory, this minimax game reaches equilibrium when the discriminator cannot distinguish real from fake images. Hence, if the discriminator is adept at distinguishing real from fake images and the generator manages to fool the discriminator, then this means that the images generated by the generator are very realistic, leading to a high-quality generative model. Then, after training a GAN, we can then feed the generator with random noise so as to generate new images (or data) that are similar to the training set.

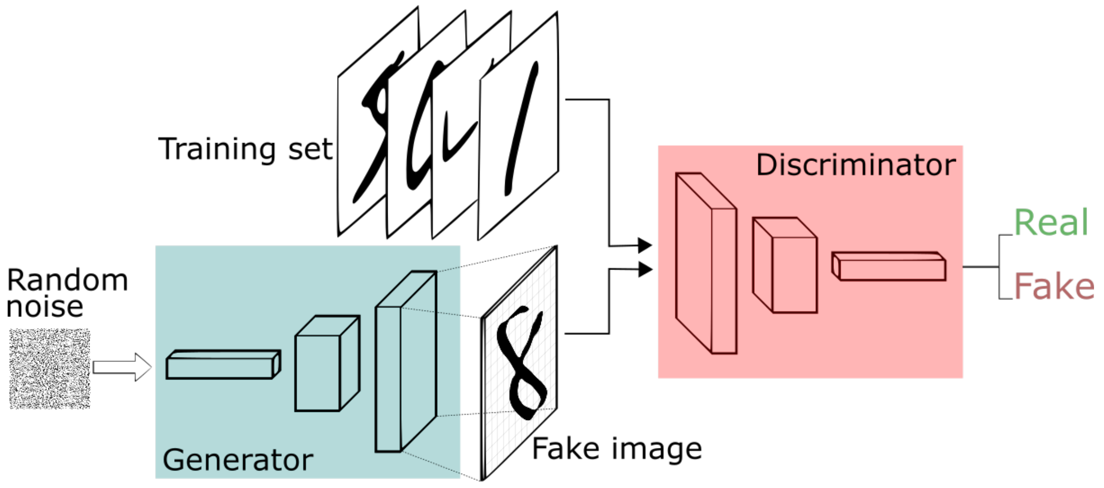


Figure 4.2.1: Generative Adversarial Network

4.2.2 Federated DP GANs

Generally GANs, just like Neural Networks, are usually trained using centralized datasets. However, as we previously mentioned, GANs were also adapted to a Differentially Private Federated setting by Augenstein et al. [13], as shown in Algorithm (3). In their approach, the authors federated only the discriminator of GANs and kept a global generator that could be trained centrally using the federated discriminator. This approach is based on the fact that in GANs, only the discriminator network needs access to real data for training. In particular, training the discriminator requires fake images and real images, and the loss function is determined by how well the discriminator distinguishes real from fake images. On the other hand, training the generator requires supplying random noise into the generator in order to generate fake images. Those images are then passed through the discriminator which outputs a prediction of whether they are real or fake. Then, the generator uses those predictions and the fake images to calculate how much it managed to fool the discriminator, which determines the value of loss function.

Hence, it is evident that access to real data is required only during the training of the discriminator, which is why the authors federated only the discriminator. Additionally, the authors enforce user-level DP on the federated training of the discriminator. Hence, since the discriminator is Differentially Private, the Generator which is trained only through the discriminator follows the same DP guarantees as the discriminator (DP post-processing lemma). Then, the authors experimented with differential privacy and utility using a small population size, and estimated the level of privacy they would achieve in realistic population sizes.

This approach made us wonder if we could federate VAEs instead of GANs under the presence of Differential Privacy. In fact, the authors, in the Open Problems section, wonder if it would be possible to federate VAEs using differential privacy, while keeping a private encoder for every user and a global (federated) decoder that will act as a data generator for all users. This is exactly what we did, as we will discuss in the next section.

However, our approach differs from the Federated GAN approach in several ways. First and foremost, we are interested in conditional sample generation, meaning that we want to be able to control which label to generate samples from. This requirement stems from the fact that our end goal is to generate an artificial private dataset which has both data and labels. Secondly, we use a different metric to assess the quality of the generated data. In particular, we use the accuracy of a neural network that is trained on artificial data and tested on the real data in order to assess the quality of the synthetic datasets. Last but not least, apart from Central Differential Privacy that the authors studied, we also experimented with Local Differential Privacy which is a stronger version of privacy which doesn't require the assumption of a trusted server/aggregator.

In any case, we will later use DpFedGANs as a comparison to challenge our Federated VAEs. However, in order to do that, we have to be able to choose which class our GAN generates samples from. This was done by training 1 federated GAN for each data class i, where the i-th GAN was trained only using the subset of the data samples that corresponds to i-th class.

Algorithm 3: DP-FedAvg-GAN

```
parameters: round participation fraction $q \in (0, q]$, total number of users $N \in \mathbb{N}$,
  total number of rounds $T \in \mathbb{N}$, noise scale $z \in \mathbb{R}^+$, gradient clipping parameter
  $S \in \mathbb{R}^+$

Initialize generator $\theta_G^0$, discriminator $\theta_D^0$,privacy accountant $\mathcal{M}$

Set $\sigma = \frac{zS}{qN}$
foreach round $t \in \{1, 2, \dots, T\}$ do
    $C^t \leftarrow$ (sample of qN distinct users)
    foreach client $k \in C^t$ do
        $\Delta_k^{t+1} \leftarrow$ UserDiscUpdate($k, \theta_D^t, \theta_G^t$)
    end
    $\Delta^{t+1} = \frac{1}{qN} \sum_{k \in C^t} \Delta_k^{t+1}$
    $\theta_D^{t+1} \leftarrow \theta_d^t + \Delta^{t+1} + \mathcal{N}(0, I\sigma^2)$
    $\mathcal{M}$.acum_privacy_spending($z$) $\theta_G^{t+1}$ $\leftarrow$GenUpdate($\theta_D^{t+1}, \theta_G^t$)
end
print $\mathcal{M}$.get_privacy_spent()

Function UserDiscUpdate($k, \theta_D^0, \theta_G$):
    parameters: number of steps $n \in \mathbb{N}$, Batch size $B \in \mathbb{N}$, learning rate $\eta_D \in \mathbb{R}^+$,
      client id $k$, clipping parameter $S \in \mathbb{R}^+$, generator input size $n_U$, gen.
      function $G(U; \theta_G)$,disc. loss function $l_D(\theta_D; b_{\text{real}}, b_{\text{fake}})$

    $\theta_D \leftarrow \theta_D^0$
    $\mathcal{B}$ $\leftarrow$(k's data split into $n$ size $B$ batches)
    foreach batch $b_{real} \in \mathcal{B}$ do
        $U \leftarrow$ (sample $B$ random vectors of dim $n_U$ )
        $b_{\text{fake}} \leftarrow G(U; \theta_G)$//Generated data
        $\theta_D \leftarrow \theta_D - \eta_D \nabla l_D(\theta_d; b\text{real}, b\text{fake})$
    end
    $\Delta = \theta_D - \theta_D^0$
    return $\Delta_k = \Delta \cdot \min\left(1, \frac{S}{\|D\|}\right)$
End Function
Function GenUpdate($\theta_D, \theta_G^0$):
    parameters: number of steps $n \in \mathbb{N}$, Batch size $B \in \mathbb{N}$, learning rate $\eta_G \in \mathbb{R}^+$,
      generator input size $n_U$, gen. function $G(U; \theta_G)$, gen. loss function
      $l_G(\theta_G; b, \theta_D)$

    $\theta_D \leftarrow \theta_D^0$
    foreach generator training step i fromm 1 to $n$ do
        $U \leftarrow$ (sample $B$ random vectors of dim $n_U$ )
        $b_{\text{fake}} \leftarrow G(U; \theta_G)$//Generated data
        $\theta_G \leftarrow \theta_G - \eta_G \nabla l_G(\theta_G; b_{\text{fake}}, \theta_D)$
    end
    $\Delta = \theta_D - \theta_D^0$
    return $\theta_G$

End Function
```

4.3 Introduction to Variational Autoencoders

Before we discuss Variational Autoencoders, we first have to understand traditional, non-probabilistic autoencoders and see what makes them unsuitable for data generation. Then, we will examine the probabilistic modifications of Autoencoders that enable Variational Autoencoders to generate high-quality synthetic data.

4.3.1 Autoencoder Neural Networks

Generally speaking, an autoencoder is a type of Artificial Neural Network that is used to compress information in an unsupervised manner [35]. The goal of autoencoders is to learn a representation (i.e. encoding) of a set of data points, typically for dimensionality reduction and noise removal. For instance, in a highly influential publication on Science made by Hinto et. al [32], autoencoders were proposed as a very efficient method of reducing the dimensionality of data, demonstrating better results than conventional dimensionality reduction methods like PCA.

In their simplest form, autoencoders are feedforward, non-recurrent neural networks that consist of 2 connected networks: An **encoder** and a **decoder**. The encoder network takes an input $\mathbf{x} \in \mathbb{R}^n$ and compresses it into a smaller (latent) representation $\mathbf{z} \in \mathbb{R}^d$, which then the decoder uses to produce an output $\mathbf{x}'$ that is as close as possible to the original input $\mathbf{x}$. Hence, due to the fact that the purpose of the autoencoder is not to predict labels, but to reconstruct its input $\mathbf{x}$, autoencoders fall into the category of unsupervised models (i.e. don't require labels for training).

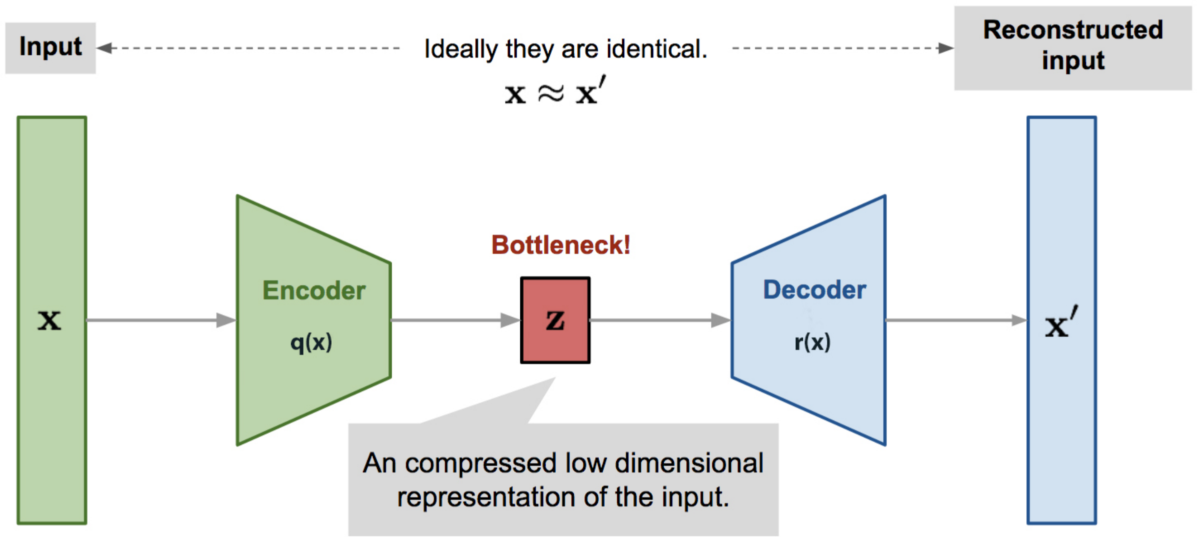

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
q(x)
Bottleneck!
$\mathbf{z}$
Decoder
r(x)
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 4.3.1: Autoencoder

The conventional autoencoder we described can be shown in the figure (4.3.1). However, autoencoders don't always take vectors as input. Convolutional autoencoders are a special type of autoencoders that operate on images instead of vectors. Their encoder consists of a CNN network that takes an image and encodes it into a vector, and their decoder is a reverse CNN network that takes that vector and tries to reconstruct the original image. For instance, in Figure (4.3.2) we can see a Convolutional Autoencoder which operates on the MNIST dataset: The encoder accepts an image of size 28×28, encodes it into a vector of size 2, and then the decoder uses that vector to reconstruct the original 28×28 image.

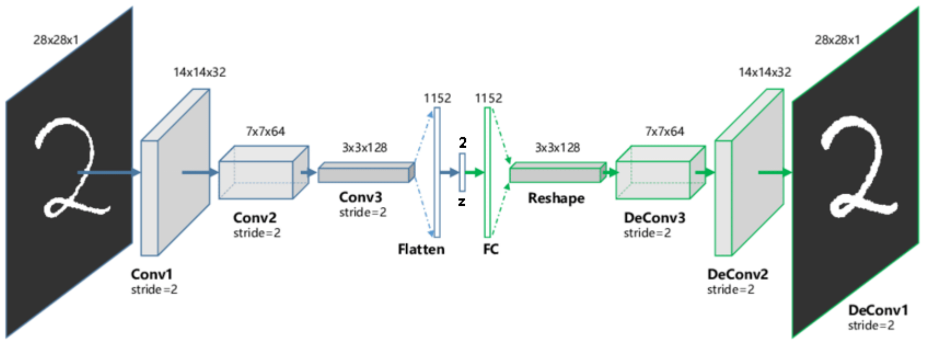


Figure 4.3.2: Convolutional Autoencoder

Mathematical Framework

In order to better understand autoencoders, it would be very useful to formally define what an autoencoder network tries to achieve. Let's assume that we have a set of training examples $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ where $\mathbf{x}_i \in \mathbb{R}^n$. Then, an autoencoder can be represented as the composition of 2 functions q and p such that [15]:

$$q : \mathbb{R}^n \to \mathbb{R}^d \quad \text{(encoder)}$$
$$r : \mathbb{R}^d \to \mathbb{R}^n \quad \text{(decoder)}$$
$$q, r = \underset{q,r}{\operatorname{argmin}} \sum_{i=1}^{m} \|\mathbf{x}_i - (q \circ r)(\mathbf{x}_i)\|^2$$

In other words, an autoencoder is trained to minimize the reconstruction error:

$$L = \sum_{i=1}^{m} \mathcal{L}(\mathbf{x}_i, \mathbf{x}'_i) \tag{4.3.1}$$

where $\mathcal{L}(\mathbf{x}, \mathbf{x}')$ usually represents the squared error between input vector $\mathbf{x}$ and output vector $\mathbf{x}' = (q \circ r)(\mathbf{x}_i)$. So, in most cases:

$$\mathcal{L}(\mathbf{x}_i, \mathbf{x}'_i) = \|\mathbf{x}_i - \mathbf{x}'_i\|^2$$

Using the loss function (4.3.1), autoencoders can then be trained through back-propagation of the reconstruction error, just like conventional feedforward neural networks.

Can Autoencoders generate?

Since our task requires a model that performs synthetic data generation, we may wonder if Autoencoders are somehow suitable for that use. As we described above, the decoder generally learns a mapping from the low-dimensional latent space to the high-dimensional output space. This enables the decoder to take as input a low dimensional vector and produce an output that -under certain assumptions- is very similar to the samples of the training set. Hence, one idea for data generation would be to supply random noise to the decoder's input, hoping that the samples will generate vectors similar to those drawn from the original dataset. Unfortunately, the latent space of a traditional autoencoder is not continuous and thus feeding the decoder with random noise is not guaranteed to generate meaningful reconstructed outputs.

For instance, let's examine the scenario of the convolutional autoencoder of Figure (4.3.2) that operates on the MNIST dataset. In that setting, our goal would be to use the trained autoencoder to generate synthetic images, similar to the ones of the original dataset. However, if we plot the 2D latent representations of the different images of MNIST after passing those images from the encoder, the plot will look like Figure (4.3.3), where different colors represent the different classes of MNIST. Hence, as we can see, there are many discontinuities between the clusters of the 2D latent space and if we choose a point that lies in those discontinuities and feed it as input to the decoder, then the generated image will be unrealistic, as the decoder doesn't know how to deal with those points.

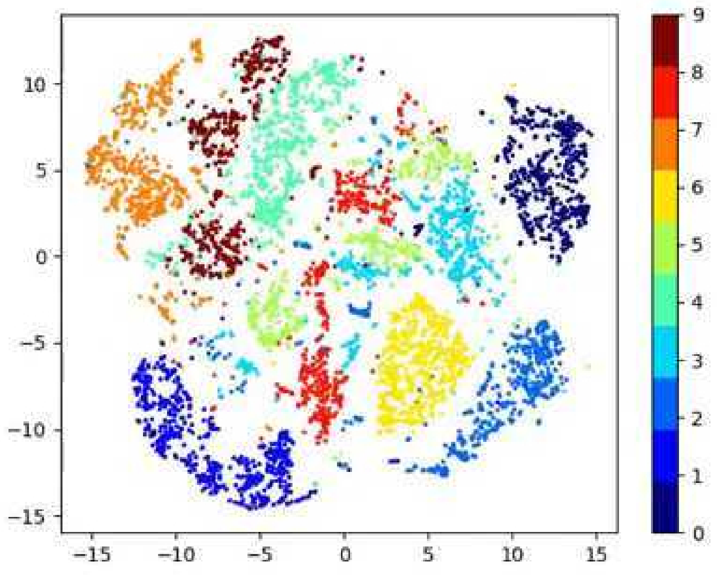


Figure 4.3.3: Autoencoder 2D latent space discontinuity (Source: [5])

4.3.2 Variational Autoencoders

In order to address the discontinuities and irregularities in the latent space of conventional Autoencoders, Variational Autoencoders (VAEs) were developed. The fundamental difference between those two models is that the latent space of VAEs is, by design, continuous, thus allowing for easy random sampling and interpolation. In other words, a variational autoencoder is an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties for data generation [22].

Just like traditional Autoencoders, VAEs are also trained to minimize the reconstruction loss between their input and their output. However, in order to introduce regularization into the training of VAEs, the following modification is made: Istead of encoding VAE's input as a deterministic latent vector, we encode it as a distribution over the latent space, and in particular as a vector of means and a vector of standard deviations of a multivariate Gaussian distribution. The schematic of a Variational Autoencoder is shown in Figure (4.3.4).

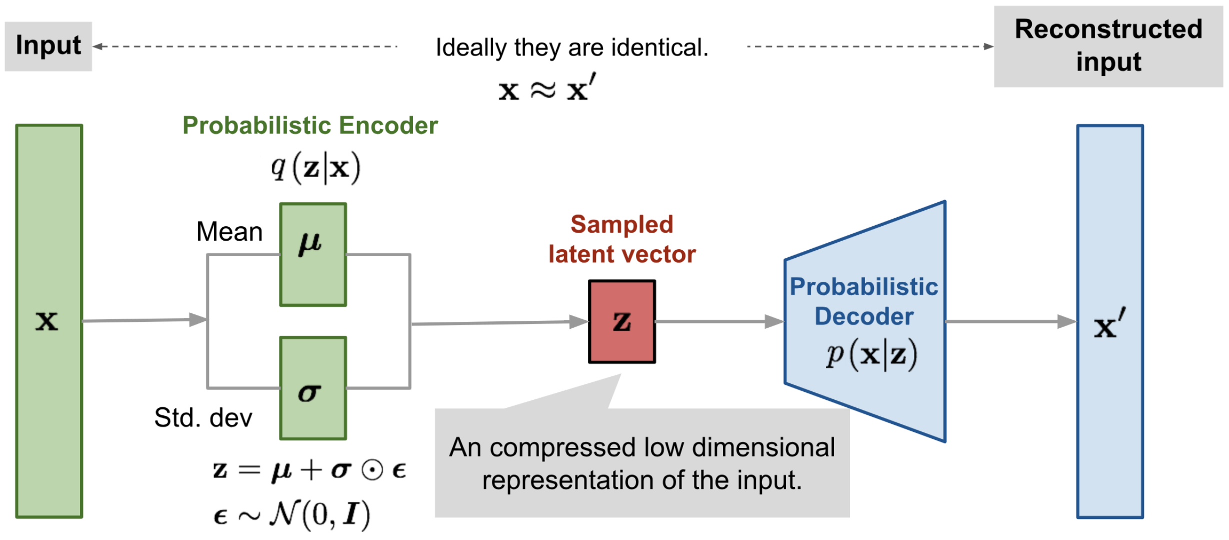

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{x}$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Figure 4.3.4: Variational Autoencoder

Intuitively, the mean vector determines where the encoded version of an input will be centered around, while the standard deviation vector controls how much the encoded point can deviate from the mean point, as seen of Figure (4.3.5). Hence, due to the fact that a given input does not correspond to a specific point but rather to a family of points around the mean, the decoder learns to generate meaningful reconstruction for all those points. This enables the decoder to maintain a relatively continuous latent space, as seen in Figure (4.3.6).

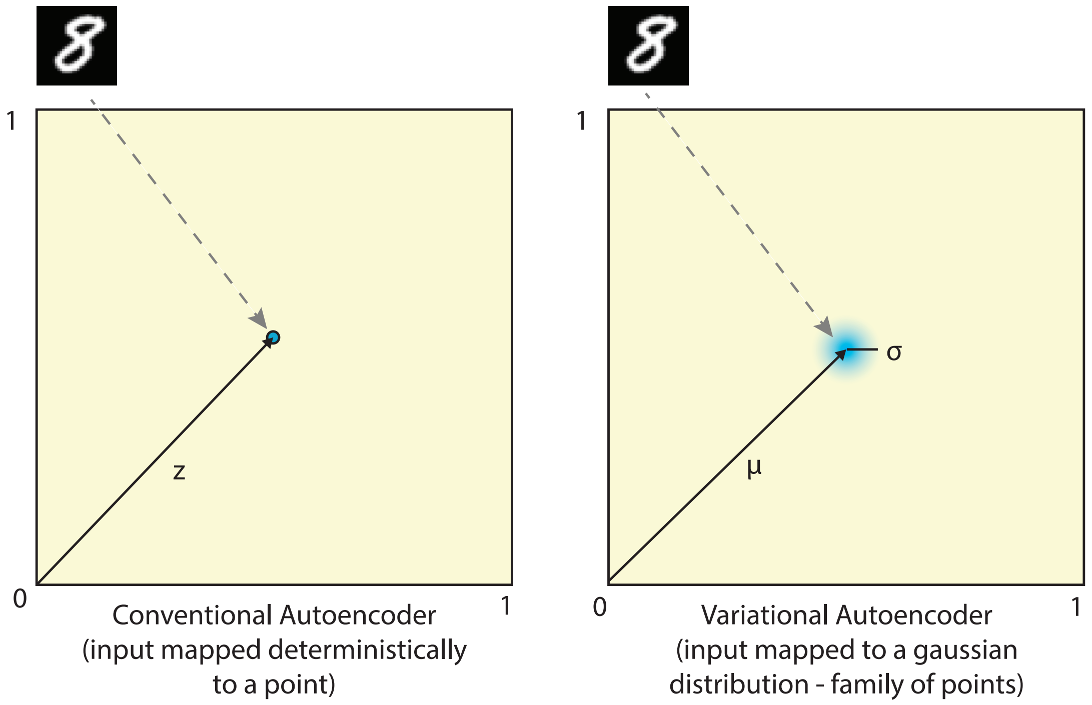

1
z
0
1
Conventional Autoencoder
(input mapped deterministically
to a point)
1
σ
μ
0
1
Variational Autoencoder
(input mapped to a gaussian
distribution - family of points)

Figure 4.3.5: Deterministic vs Stohastic mapping of input into 2D latent space

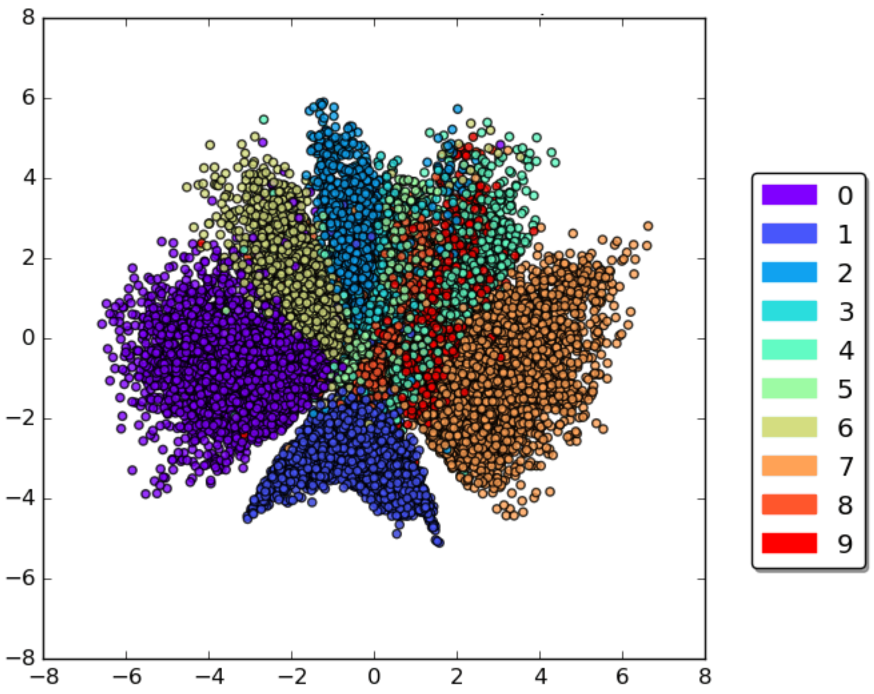


Figure 4.3.6: Continuous 2D latent space of MNIST VAE (Source: [6])

Probabilistic Framework

We will now define the mathematical framework behind VAEs, as proposed in [34]. Let's assume that we have a set of training examples $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ that are IID samples of some continuous or discrete distribution. In the context of VAEs, given an input $\mathbf{x}$, we want to map that input not to a fixed vector, but rather to a distribution. So, let's denote this distribution as $p_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$. In order to fully describe the probabilistic relationship between the input vector $\mathbf{x}$ and the latent vector $\mathbf{z}$ within the context of a VAE, we need the following parametric families of distributions[1]:

- Prior parametric distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$
- Likelihood parametric distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
- Posterior parametric distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$

If we now assume that we know the value of the parameter $\boldsymbol{\theta} = \boldsymbol{\theta}^*$, then in order to generate a sample that is similar to a real data point $\mathbf{x}_i$, we have to do the following process:

1. Sample a vector $\mathbf{z}_i$ from the prior distribution $p_{\boldsymbol{\theta}^*}(\mathbf{z})$
2. Generate a value $\mathbf{x}_i$ from the conditional distribution $p_{\boldsymbol{\theta}^*}(\mathbf{x}|\mathbf{z} = \mathbf{z}_i)$

Clearly, the optimal value $\boldsymbol{\theta}^*$ for the parameter $\boldsymbol{\theta}$ is the one that maximizes the probability of generating the samples of the original dataset. Hence, since we assumed that the

[1]When we say that $p_{\boldsymbol{\theta}}$ is a parametric family of distributions parametrized by $\boldsymbol{\theta}$ we mean that $p_{\boldsymbol{\theta}}$ may for instance be a Gaussian distribution with means and variances determined by $\boldsymbol{\theta}$, or an other type of general distribution (e.g. Poisson, Laplacian) with parameters determined by $\boldsymbol{\theta}$

samples of our dataset are IID then:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{m} p_{\boldsymbol{\theta}}(\mathbf{x}_i) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i)\end{aligned} \tag{4.3.2}$$

Also, according to the law of conditional probabilities:

$$\begin{aligned}p_{\boldsymbol{\theta}}(\mathbf{x}_i) &= \int p_{\boldsymbol{\theta}}(\mathbf{x}_i, \mathbf{z}) d\mathbf{z} = \\ &= \int p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) d\mathbf{z}\end{aligned} \tag{4.3.3}$$

Hence, in order to calculate $\boldsymbol{\theta}^*$ we could plug Equation (4.3.3) into (4.3.2). Unfortunately, the problem of calculating $p_{\boldsymbol{\theta}}$ directly is intractable, as it is very computationally expensive to iterate over all values of $\mathbf{z}$ and calculate the integral in Equation (4.3.3). This means that the posterior:

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) / p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{4.3.4}$$

is also intractable. For this reason, we will use an approximation function $q_{\phi}(\mathbf{z}|\mathbf{x})$ parametrized by ϕ to approximate the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$.

The Probabilistic Graphical Model of the scenario we described can be shown on Figure (4.3.7). In this figure, solid lines are used for the generative distribution $p_{\boldsymbol{\theta}}$ and dashed lines refer to the distribution $q_{\phi}(\mathbf{z}|\mathbf{x})$ that is used to approximate the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Hence, in that setting, the conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ represents a probabilistic decoder, as it defines a generative model that takes a latent vector $\mathbf{z}$ and produces an output sample $\mathbf{x}$. Also, the approximation function $q_{\phi}(\mathbf{z}|\mathbf{x})$ represents the probabilistic encoder, as it takes an input sample and outputs the latent vector that corresponds to that sample.

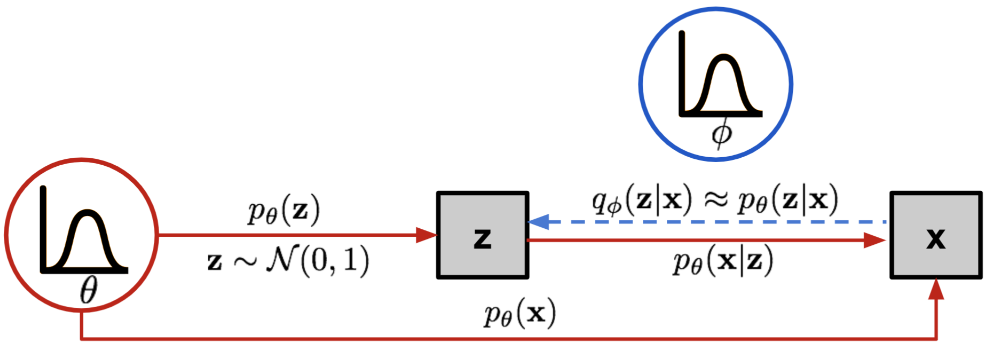

ϕ
θ
$p_\theta(\mathbf{z})$
$\mathbf{z} \sim \mathcal{N}(0,1)$
$\mathbf{z}$
$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}$
$p_\theta(\mathbf{x})$

Figure 4.3.7: Probabilistic Graphical Model of VAEs (Source: [4])

Since we introduced a function approximation $q_{\phi}(\mathbf{z}|\mathbf{x})$ of the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, we have to ensure that those functions/distributions are very close to one another. Hence, in order to measure the similarity of those distributions, we will use the Kullback-Leibler divergence. KL divergence $D_{KL}(X||Y)$ between 2 distributions X and Y measures how different distribution Y is in comparison to the reference distribution X. In particular, it measures the amount of information (in nats or 1/log(2) bits) that we have to use in order to distort distribution Y into X.

Within the context of Variational Autoencoders, we want to minimize the inverse KL Divergence $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$ with respect to $\boldsymbol{\phi}$ in order to ensure that $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ approximates well the posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Hence, by expanding the KL divergence term, we get:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z}$$

And since $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{z},\mathbf{x})/p_{\boldsymbol{\theta}}(\mathbf{x})$, then:

$$\begin{aligned} D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \\ &= \int_z q_\phi(\mathbf{z}|\mathbf{x}) \big(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} \big) d\mathbf{z} \end{aligned}$$

Also, due to the fact that $\int_z q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} = 1$ then:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z}$$

Finally, due to the fact that $p(z,x) = p(x|z)p(z)$ we get:

$$\begin{aligned} D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) &= \log p_\theta(\mathbf{x}) + \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \\ &= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} [\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \\ &= \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) \end{aligned}$$

By rearranging the terms in the equation above we get:

$$\begin{aligned} &\log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \end{aligned} \tag{4.3.5}$$

The LHS of the last equation is precisely what we are trying to maximize in order to learn the true distributions. Essentially, in order to train our variational model, we want 2 things:

1. Maximize the likelihood (or log-likelihood) of generating real data. This can be done by maximizing the term $\log p_{\boldsymbol{\theta}}(\mathbf{x})$

2. Minimize the difference between the real posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ and its approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. This can be done by minimizing the KL term $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$ or equivalently maximizing the negative of that KL term.

Hence, in order to train our Variational Autoencoder, we have to maximize the LHS of (4.3.5), which gives us the following loss function:

$$\begin{aligned} \mathcal{L}_{VAE}(\boldsymbol{\theta},\boldsymbol{\phi}) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \end{aligned} \tag{4.3.6}$$

and thus if we denote the optimal parameters as $\boldsymbol{\theta}^*$ and $\boldsymbol{\phi}^*$ then:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \underset{\boldsymbol{\theta},\boldsymbol{\phi}}{\operatorname{argmin}} \mathcal{L}_{VAE}(\boldsymbol{\theta},\boldsymbol{\phi})$$

In Variational Bayesian settings, the loss function (4.3.6) is usually referred to as the Variational Lower Bound or Evidence Lower Bound. The terminology "Lower Bound" stems from the fact that since the KL divergence term is always non negative, the following inequality holds:

$$\begin{aligned} -\mathcal{L}_{VAE}(\boldsymbol{\theta}, \phi) &= \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\ &\leq \log p_\theta(\mathbf{x}) \end{aligned}$$

Consequently, by minimizing the loss function (4.3.6) we are maximizing the lower bound of the probability that our Variational Autoencoder generates real samples.

Evaluating the loss function

Since we have defined the loss function and we have made the optimization problem tractable, we should now focus on how we can evaluate the loss function (4.3.6) which is essential to train the Variational Autoencoder. In order to do that, we will make the following assumptions which we will use throughout this work:

- The distribution $p_{\boldsymbol{\theta}}(\mathbf{z})$ where $\mathbf{z} \in \mathbb{R}^d$ is a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ with:

$$p_{\boldsymbol{\theta}}(\mathbf{z}) = \frac{1}{\sqrt{(2\pi)}^d} \exp\left(-\frac{1}{2}\|\mathbf{z}\|^2\right) \tag{4.3.7}$$

- The distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \mathrm{diag}(\boldsymbol{\sigma}_\mathbf{x})^2)$ with:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \frac{1}{\sqrt{(2\pi)}^d \prod_{i=1}^d \sigma_\mathbf{x}^{(i)}} \exp\left(-\frac{1}{2}\sum_{i=1}^d \left(\frac{z_i - \mu_\mathbf{x}^{(i)}}{\sigma_\mathbf{x}^{(i)}}\right)^2\right) \tag{4.3.8}$$

  where $\boldsymbol{\mu}_\mathbf{x}$ and $\boldsymbol{\sigma}_\mathbf{x}$ is the vector of means and the vector of standard deviations that is produced if we give the encoder $\mathbf{x}$ as input (see figure (4.3.4)).

Before proceeding, we will present the following lemma which allows us to calculate the KL divergence between 2 Gaussian distributions:

Lemma 4.1 *If P_1 and P_2 are gaussian distributions with means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbb{R}^d$ and standard deviations $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2 \in \mathbb{R}^d$ then:*

$$D_{KL}(P_1||P_2) = \frac{1}{2}\left(\log\frac{|\boldsymbol{\Sigma}_2|}{|\boldsymbol{\Sigma}_1|} - d + tr(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)\right)$$

Using the lemma (4.1) and since $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \mathrm{diag}(\boldsymbol{\sigma}_\mathbf{x})^2)$ we can calculate the KL divergence that appears in the loss function as:

$$\begin{aligned} D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) &= \frac{1}{2}\left(\log\frac{1}{\prod_{i=1}^d \sigma_\mathbf{x}^{(i)}} - d + \|\boldsymbol{\sigma}_\mathbf{x}\|^2 + \|\boldsymbol{\mu}_\mathbf{x}\|^2\right) = \\ &= \frac{1}{2}\left(\|\boldsymbol{\sigma}_\mathbf{x}\|^2 + \|\boldsymbol{\mu}_\mathbf{x}\|^2 - d - \prod_{i=1}^d \sigma_\mathbf{x}^{(i)}\right) \end{aligned} \tag{4.3.9}$$

Also, the term $\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})$ of the loss function can be calculated by drawing random samples $\mathbf{z}$ from $\mathcal{N}(\boldsymbol{\mu}_\mathbf{x}, \mathrm{diag}(\boldsymbol{\sigma}_\mathbf{x})^2)$ for every training example $\mathbf{x}$, and then passing $\mathbf{z}$ through the decoder. Hence, using the above term and the KL term, we can calculate (or to be more precise, approximate) the loss function.

However, apart from this method, there is also an approximate method to estimate the loss function, which is used instead in many practical cases. This method leverages the fact that:

$$\begin{aligned}
\text{ELBO} = -\mathcal{L}_{VAE}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \\
&= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) = \\
&= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \Big[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log \frac{p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \Big] = \\
&= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \Big[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log p_{\boldsymbol{\theta}}(\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \Big]
\end{aligned}$$

and then uses samples $z \sim q_\phi(\mathbf{z}|\mathbf{x})$ to produce a Monte-Carlo approximation of the expectation. In particular, the 3 terms inside the expectation can be approximated as follows:

1. The term $\log p_{\boldsymbol{\theta}}(\mathbf{z})$ can be approximated by taking the logarithm of the PDF of a standard normal multivariate Gaussian $\mathcal{N}(0, \mathbf{I})$.
2. The term $\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can be approximated by taking the logarithm of the PDF of a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}}, \mathrm{diag}(\boldsymbol{\sigma}_{\mathbf{x}})^2)$.
3. The term $\log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ (since the distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is unknown) can be estimated through an approximation of cross entropy by comparing the output of the decoder with the input of the encoder.

the term $\log p_{\boldsymbol{\theta}}(\mathbf{z})$ can be approximated by evaluating the PDF of standard gaussian $\mathcal{N}(0, \mathbf{I})$

Reparameterization Trick

As we described above, evaluating the expectation term $\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})$ requires generating samples $\mathbf{z}$ from the distribution $q_\phi(\mathbf{z}|\mathbf{x})$. Unfortunately, the sampling operation is a stochastic process and thus doesn't allow the backpropagation of the gradients. Hence, since backpropagation is an integral part of Neural Network training, we have to do something in order to remove the stochasticity from the training process. For this reason, we will express the random variable $\mathbf{z}$ as a deterministic variable $\mathbf{z} = g(\boldsymbol{\phi}, \mathbf{x}, \varepsilon)$ where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is an auxiliary random variable and g is a transformation function parametrized by $\boldsymbol{\phi}$ which converts ε to $\mathbf{z}$.

In particular, before this reparametrization, the sampling operation was:

$$\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}}, \mathrm{diag}(\boldsymbol{\sigma}_{\mathbf{x}})^2)$$

and after the reparametrization, the sampling operation becomes:

$$\mathbf{z} = \boldsymbol{\mu}_{\mathbf{x}} + \boldsymbol{\sigma}_{\mathbf{x}} \odot \varepsilon, \text{ where } \varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where $\mathbf{z}, \boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\sigma}_{\mathbf{x}}, \varepsilon \in \mathbb{R}^d$ and the vectors $\boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\sigma}_{\mathbf{x}}$ are the outputs of the encoder when given an input $\mathbf{x} \in \mathbb{R}^n$.

Using this reparameterization trick, $\mathbf{z}$ now becomes a deterministic node and the stochasticity of sampling is transferred to the random variable ε. This enables us to backpropagate the losses through the Autoencoder during training, as seen on Figure (4.3.8).

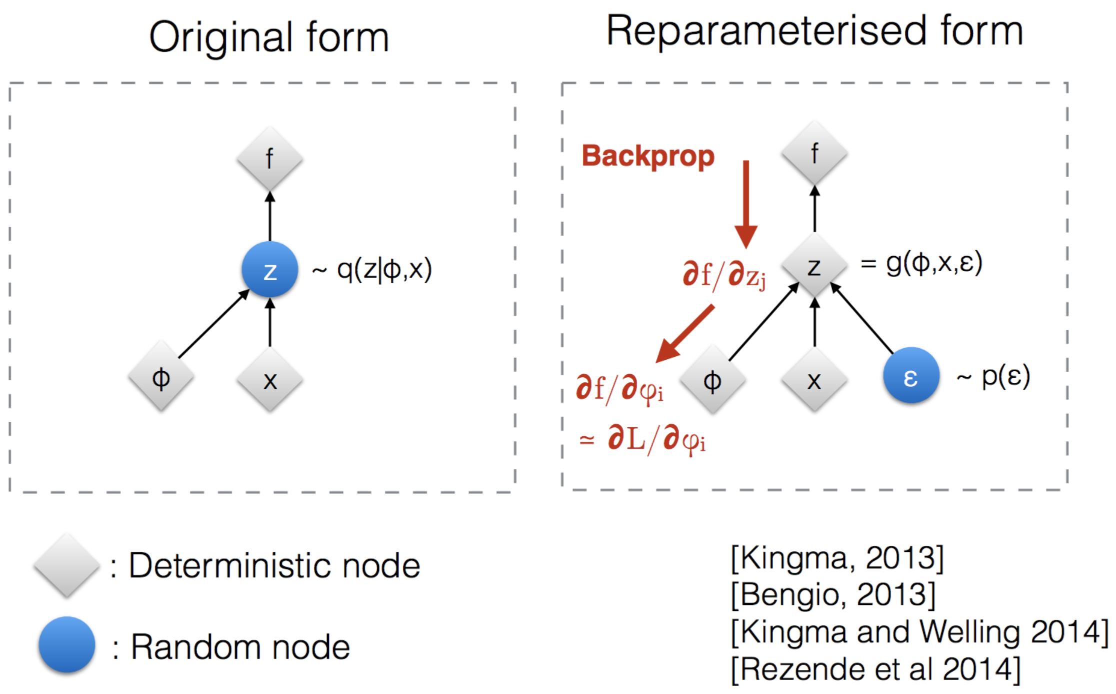


Figure 4.3.8: Reparameterization (Source: NIPS 2016 workshop)

Training

After analyzing the reparametrization trick, we can now describe the training process of a Variational Autoencoder:

1. An input sample $\mathbf{x}$ is fed into the encoder which converts it into the parameters of a multidimensional gaussian distribution. In particular, the output of the encoder is a vector of means $\boldsymbol{\mu}_x$ and a vector of standard deviations $\boldsymbol{\sigma}_x$ as shown on figure (4.3.4).

2. A sample $\mathbf{z}$ from the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_x, \text{diag}(\boldsymbol{\sigma}_x)^2)$ is generated.

3. The sampled point is fed into the decoder and the ELBO error is calculated.

4. The reconstruction loss is backpropagated through the network

It should be noted that this process is usually done on batches of training examples and not on individual training samples in order to increase the training speed and improve the fidelity of the model.

In any case, by performing the steps we described over many epochs, we are able to obtain a trained, Variational Autoencoder that is capable of data generation. In particular, if we now feed the decoder with random noise $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, then the samples generated in the output of the decoder will be similar to the ones of our training set. This happens because the latent space of a VAE is continuous, thus enabling sampling and interpolation of the latent space.

4.4 Federated Variational Autoencoders

So far, we have explained in detail how Variational Autoencoders work. Hence, we will now attempt to train them in a federated setting, while ensuring meaningful guarantees of Differential Privacy. For that reason, we will first describe our target federated scenario and then explain our methodology for training a VAE under that setting, while measuring the privacy cost that the federated training incurs.

4.4.1 Federated Setting

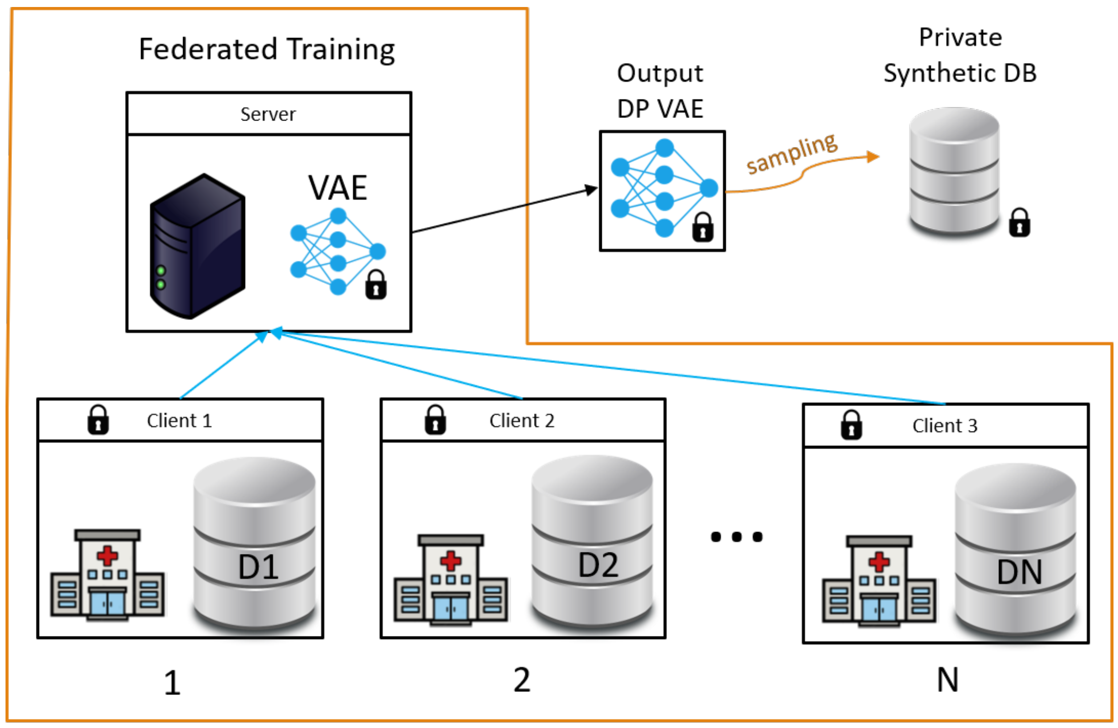


Figure 4.4.1: Federated VAE setting

In our federated setting, we will assume that there are N different clients (such as hospitals), where each one of those clients has his own private dataset D_i. The goal of the clients is to collaboratively train a VAE model using their collective data, but with the added constraint that the private datasets never leave the client and that the training is done under user-level differential privacy. This requirement for differential privacy stems from the fact that the clients own sensitive datasets and thus want to ensure that the training process respects their privacy. Also, the requirement for user-level differential privacy instead of example-level differential privacy is of paramount importance, as we want to provide privacy protections to an entire client and not just on a subset of the client's data.

In order for the federated training to happen, there is a centralized server which aggregates the local model updates of the clients. The server then uses those aggregates to update a centralized version of the VAE model, which is sent to the clients at the beginning of every federated training epoch. At the same time, the server also keeps track of the privacy loss during training. Then, after the training finishes, the VAE model that lies in the server can be safely released to the public for synthetic data generation, without the fear of it compromising the privacy of data owners. This process is illustrated in figure (4.4.1).

To trust or not to trust?

One of the fundamental questions we have to answer is whether or not the clients trust the server to aggregate their local updates. If the clients are certain about the server's credibility, then they may entrust the server with the task of implementing Differential Privacy. This means that the clients will send a clipped but non-noisy version of their local updates to the server, and then the server will aggregate the clipped local updates while adding appropriate noise. This privacy model is the Central Differential Privacy and makes the strong assumption that the server is fully trusted. The benefit of Central Differential Privacy is that since the parties trust one another, there is a great benefit in utility, as the clients don't need to add more noise than necessary to ensure privacy.

Unfortunately, in many cases, relying on a trusted server may not always be possible. For instance, if the server becomes compromised, then the privacy implications may be enormous. Additionally, there is also the possibility that the server is managed by an honest-but-curious entity, which doesn't interfere with the training protocol, but instead tries to learn information about the clients. In those cases, there are 2 main approaches to alleviate the privacy risk:

1. **Combine Central DP with a secure aggregation protocol, such as MPC**. This way, the server will only have access to an aggregate (e.g. average) of the local model updates and not on the individual updates themselves. Hence, if a sufficient number of clients participate in each federated round the server won't be able to infer information about any client's data. The benefit of this approach is that it does not incur any cost on utility, although it increases the computational and communication cost of federated training.

2. **Use Local Differential Privacy instead of Central Differential Privacy**:
   In this approach, the clients clip their weights and introduce noise prior to sending the weight updates to the server. The benefit of this approach is that the clients can choose individually how much privacy they require and then add the appropriate amount of noise in order to achieve this level of privacy. Unfortunately, this approach usually comes at a very high cost on utility, as in order to attain good privacy protection in that manner, the amount of noise needed may have a significant impact on the quality of the trained model.

In our experiments, we we will assume that the clients don't trust the server. Hence, whenever we use Central Differential Privacy, we will assume that we have access to a secure aggregation protocol, such as an MPC protocol (e.g. [26]). However, since having access to such protocols may not always be possible, we will also experiment with Local Differential Privacy which doesn't make any such assumptions. Unfortunately, as we will see, Local Differential Privacy provides relatively poor privacy protection if we want to maintain acceptable accuracy levels.

Notation

Before we proceed any further, we will present the notation that we will use throughout this chapter:

- Fraction of clients participating in each round: $q \in (0, 1]$
- Total number of clients: $N \in \mathbb{N}$
- Total number of federated rounds: $T \in \mathbb{N}$
- DP noise scale: $z \in \mathbb{R}^+$,
- Gradient clipping parameter: $S \in \mathbb{R}^+$
- Client id: k

- Dataset of client k: D_k
- Batch size: $B \in \mathbb{N}$
- Learning rate: $\eta \in \mathbb{R}^+$
- Number of classification classes: $C \in \mathbb{N}$
- Size of VAE latent dim: $n_U \in \mathbb{N}$
- Encoder parameters Φ, decoder parameters Θ
- Encoder: $E(V;\Phi)$
- Decoder: $D(U;\Theta)$
- Loss function: $l(x,\hat{x};\Phi,\Theta)$

Datasets

In our federated scenario, we have made the assumption that there is a dataset D which is distributed among N different clients and each client i has its own private dataset D_i. Each private dataset can be described collection of n_i training examples and their corresponding labels (i.e. $D_i = \{(\mathbf{x}_i^{(1)}, y_i^{(1)}), \dots (\mathbf{x}_i^{(n_i)}, y_i^{(n_i)})\}$). However, although different clients can have different number of training examples, all training examples must have **the same number of dimensions**. Under those assumptions, after training a Federated VAE, we will be able to generate new synthetic examples and labels by forming a Differentially Private Synthetic Database that has the exact same attributes and columns as any private database D_i.

In our federated setting, we will experiment 2 different datasets, the EMNIST dataset, which is an image dataset and the Epilepsy dataset, which is a tabular dataset:

1. **EMNIST** [19] is an extension of the well known MNIST dataset. It contains a collection handwritten character digits which are derived from the NIST Special Database 19. The images are converted to a 28×28 format so that the structure of the EMNIST is exactly the same as that of MNIST. Although EMNIST contains both letters and numbers, we will only work with the portion of the dataset that contains digits the $0-9$, so that we have a manageable number of classes ($C = 10$).
2. **Epilepsy** [11] is a tabular dataset that is used to predict epileptic seizures based on brain activity. The version of the dataset that is available on the UCI Machine learning Repository [7] contains 11500 rows in total. Every row consists of 178 real-number attributes that represent the value of an EEG recording across different points in time. Although the possible labels for every row are 5, only the 1st label indicates an epileptic seizure, whereas the other labels represent other states of the brain. Hence, just like many previous authors, we will group classes $2-5$ together so as to obtain a binary classification task: A label of 1 will indicate that the patient is having an epileptic seizure, while a label 0 will indicate that they are not having a seizure. However, it should be noted that grouping together classes $2-5$ results in an **unbalanced** dataset, as 80% of the training examples correspond to label 0 and 20% correspond to label 1.

Given those centralized datasets, one fundamental question is how we can distribute them into clients. Hence, since data distribution plays a huge role in federated learning, we will experiment with the following different data distribution strategies:

- **Uniform:** In this strategy, we will distribute our dataset uniformly among clients, so that every sample has the same probability of being assigned to any given client. More formally, for every training example $(\mathbf{x}, y)$ of the original dataset D and for every client $k \in \{1, \ldots, N\}$, it must be true that:

$$Pr[(\mathbf{x}, y) \in D_k] = \frac{1}{N} \tag{4.4.1}$$

  A realization of the uniform data distribution strategy can be shown in figure (4.4.2a).

- **KMEANS:** In this strategy the data distribution was perfored using KMEANS on the features of our samples. This data distribution scheme results in clients with vastly different numbers of samples, as seen in figure (4.4.2b).

- **Geometric:** In this scheme we used a geometric distribution to determine which cluster any given sample will be assigned to. So, the probability of a sample $u = (\mathbf{x}, y)$ being assigned to client k is:

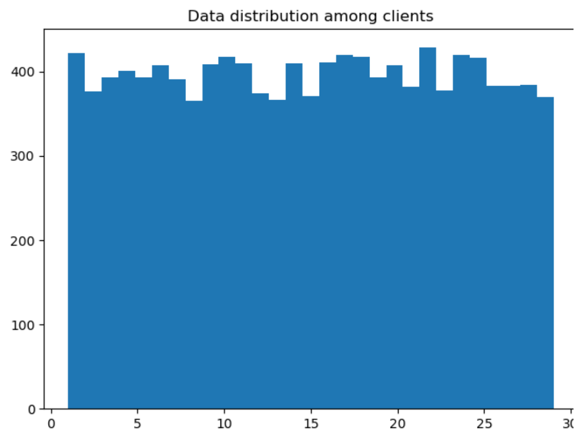
$$Pr[u \in D_k] = p(1-p)^{k-1}, \quad k \geq 1 \tag{4.4.2}$$

  If we now use $X_i \in \mathbb{N}^*$ to denote the id of the client that the ith sample was assigned to, then we want to ensure that the probability that X_i exceeds N is very small. The reason we want that probability to be small is that we want most samples to be assigned to clients with id between 1 and N inclusive, as we only have N clients. Hence, since the geometric distribution is parametrized by p, then given a probability threshold a of our choice, we need to find a value of p such that:

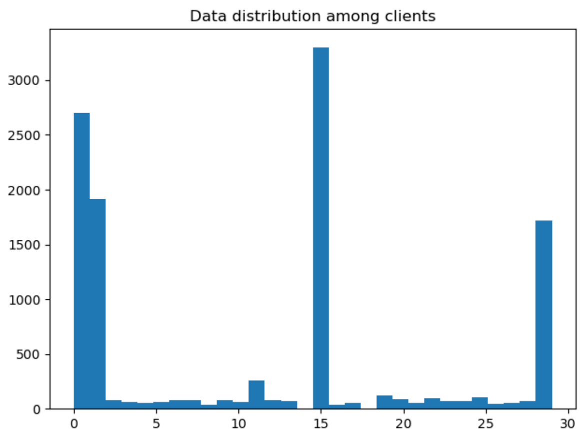
$$\begin{aligned} &P(X_i \leq N) \geq a \iff \\ &1-(1-p)^N \geq a \iff \\ &p \geq 1-(1-a)^{\frac{1}{N}} \end{aligned} \tag{4.4.3}$$

  So, if we pick $p = 1-(1-a)^{\frac{1}{N}}$, then we will know that any given sample will be assigned an id between 1 and N inclusive with probability a. For instance, if we have $N = 30$ clients and we set $a = 0.99$, then p is calculated as 0.14. Hence, this means that with a value of $p = 0.14$, the probability that a sample is assigned to a client with an id more than 30 is $1 - a = 0.01$.

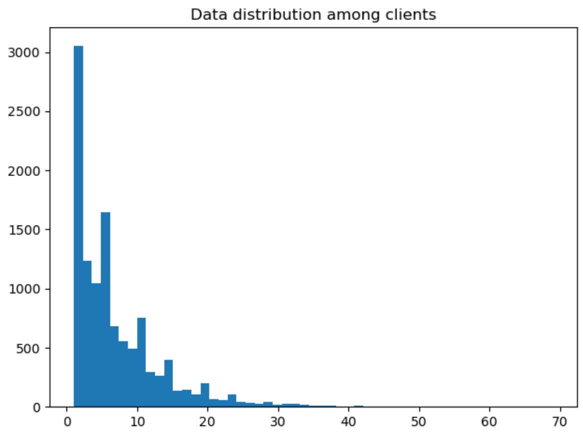
Between those different data distribution strategies, we expect that the Uniform strategy will produce the best synthetic datasets, as this strategy leads to an IID federated dataset, which usually yields good results in the training of a model. On the other hand, we expect that the KMEANS distribution strategy will have the worst results, as it leads to very non-iid client datasets with vastly different number of samples among clients.



(a) Uniform



(b) KMEANS



(c) Geometric

Figure 4.4.2: Number of samples per client in epilepsy dataset

4.4.2 Proposed method

Our task is to federated VAEs so as to create a differentially private synthetic data generator from distributed data. However, if we train a VAE model using samples $\mathbf{x}$ of the original dataset, we will be able to generate artificial samples that are similar to $\mathbf{x}$, but we won't be able to determine the corresponding labels y. Hence, apart from the individual data points $\mathbf{x}$, we also have to find a way to synthesize the dataset labels y. For this reason, we will examine 2 different approaches for this problem:

- **FedVAESep:** In this approach, we will use 1 VAE for every class (C VAEs in total). Hence, whenever we want to generate data from the i-th class, we just need to sample the decoder of the i-th VAE.
- **FedVAEUni:** In this approach, we will use 1 VAE for all classes, but we will need to make a slight modification to the decoder so that we will also be able to specify to the decoder which class we want to generate data from. Then, in order to generate data from a particular class, we will feed the decoder with a vector of random noise concatenated with the one-hot encoded version of our label.

Each one of these approaches has certain advantages and disadvantages. In particular, the FedVAESep approach, where 1 VAE is used for every class, has the advantage that it can potentially be used in skewed datasets, where there are vastly different numbers of samples in the different classes. In those cases, if we used 1 VAE for all classes, then the classes which have much more samples could dominate the training of the single VAE, and thus the VAE would not be very good at generating the minority classes. On the other hand, if we separate the classes and use 1 VAE for every class, then each VAE will be trained solely on its corresponding class, thus learning to generate this class much more efficiently.

The obvious disadvantage of the FedVAESep approach is that using 1 VAE for every class yields a much higher computation and communication overhead than using 1 VAE for all classes. In particular, in the FedVAESep approach, the clients have to train C different VAEs and communicate the parameters of those VAEs to the server, whereas in the FedVAEUni approach, the clients have to train and share the parameters of just 1 VAE. In addition to that, the FedVAEUni approach can potentially lead to better privacy guarantees, as the less information the clients have to share with the server, the better the privacy guarantees they can get. In fact, if the clients share only the parameters of 1 VAE with the server, they need to perform a much less aggressive gradient clipping than if they share the parameters of multiple VAEs. This means that FedVAEUni may be able to achieve better utility than FedVAESep if the same level of privacy is used.

Consequently, it is evident that if the dataset has good properties that allows us to use FedVAEUni, we should choose FedVAEUni over FedVAESep, due to the computation, communication and privacy benefits of FedVAEUni. On the other hand, if the dataset doesn't allow us to use FedVAEUni, we can use FedVAESep which may work better on skewed datasets, albeit with higher computation, communication and privacy cost.

If we now combine the FedVAEUni and FedVAESep approaches with Central and Local Differential privacy, we obtain the 4 different federation methods of Table (4.4.1), which we will discuss in detail below.

	1 VAE for each class	**1 VAE for all classes**
Central DP	FedVAESepCDP	FedVAEUniCDP
Local DP	FedVAESepLDP	FedVAEUniLDP

Table 4.4.1: Different VAE federation schemes

FedVAESepCDP

The main idea of this federation method is to train 1 VAE for every class, while utilizing Central Differential Privacy. This way, after the training is finished, we can sample the decoder of the i-th VAE to generate synthetic data of the i-th class. Hence, in order to train VAEs under this setting, we will follow an approach similar to [13] and [43], while utilizing Central DP. This leads us to the following training procedure:

1. The server initializes C VAEs, one for each class.
2. The server randomly selects qN clients out of the pool of N clients.
3. The server sends the current weights of the decoders of the C VAEs to the selected clients.
4. The selected clients load the weights of the decoders sent to them by the server. Then, the clients use their local samples corresponding to the i-th class to train the i-th VAE using 1 epoch of mini-batch SGD. This process is repeated for all C classes.

5. The clients calculate the weight differences Δ_i of the C decoders by subtracting the decoder weights sent by the server from the local decoder weights after training.

6. The clients clip the C weight difference vectors Δ_i so that each of those vectors has L_2 norm bounded by $S/\sqrt{C}$ and their concatenation has L_2 norm bounded by S.

7. The clients send their clipped non-noisy update vectors for each one of the C decoders to the server.

8. The server aggregates the update vectors for every decoder and adds Gaussian noise of std σ_G in order to ensure Central Differential Privacy.

9. The server updates the decoders of its C VAE models using the aggregated gradients.

10. The process continues from step 2 for $T-1$ more rounds.

11. **Data Generation**: For every class $i = \{1, \dots, C\}$ we can feed the decoder of the i-th VAE of the server with random Gaussian noise $\mathcal{N}(0, I)$ in order to generate data for class i.

This process is described in detail in Algorithm (4). However, at this point,we should note that during the federation process we described, we chose to federate **only** the decoders and not the encoders of VAEs. This means that although both the encoders and the decoders are trained locally by the clients, only the weight updates of the decoders are sent back to the server. The reason for this is that in order for the server to be able to generate synthetic data, it only needs to feed random noise to the decoders: The encoders are only needed during the training happening locally at the clients. Hence, this means that each client can have its own private encoder, without sharing it with the server. This approach has 2 main advantages:

- **Less communication overhead**: By only sending the weight updates of the decoders back to the server, the clients can significantly reduce the amount of data they have to communicate to the server. This is particularly important when the clients are low-power devices with limited bandwidth and poor connectivity, such as mobile devices.

- **Better privacy guarantees**: By sharing only the updates of the decoders, we are able to obtain better privacy guarantees for the same amount of noise. In particular, if we wanted to share the weight updates of both the encoder and the decoder, we would need to perform a more aggressive clipping on the weight differences in order to obtain the same DP guarantees. Hence, this could potentially lead to worse utility for the same level of privacy.

Let's now reason about the privacy guarantees of Algorithm (4). In order to analyze this Central DP scenario, we will assume that the clients trust the server, or that we have access to a secure aggregation protocol (e.g. MPC). Under those assumptions, the Algorithm (4) obeys user-level central differential privacy, as it follows the same privacy structure as DP-FedAvg algorithm in [43]. Also, in order to measure the impact of noise in DP, we have to examine the L_2 sensitivity of the following aggregator, which is used in Algorithm (4):

$$f_G(C^t) = \frac{\sum_{k \in C^t} \Delta_k}{qN} \tag{4.4.4}$$

So, if we use $\Delta_k^{(i)}$ to denote the weight differences of the i-th decoder of the k-th user, then clearly $\|\Delta_k^{(i)}\| \leq \frac{S}{\sqrt{C}}$ due to the clipping performed by the clients in Algorithm

(4). Hence, if we concatenate the updates of all decoders into a long vector $\Delta_k = \text{concat}(\Delta_k^{(1)}, \Delta_k^{(2)}, \ldots, \Delta_k^{(C)})$, then we have:

$$\begin{aligned} ||\Delta_k||^2 &= \sum_{i=1}^{C} ||\Delta_k^{(i)}||^2 \leq \sum_{i=1}^{C} \frac{S^2}{C} = S^2 \Longrightarrow \\ ||\Delta_k|| &\leq S \end{aligned} \tag{4.4.5}$$

Hence, since every update vector Δ_k has an L_2 norm bounded by S, then according to [43], the sensitivity of the aggregator $f(C^t)$ is:

$$\mathbb{S}(f_G) = \frac{S}{qN} \tag{4.4.6}$$

Given the sensitivity of the query estimator, if we pick $\sigma = \sqrt{2\ln(1.25/\delta)} \cdot \mathbb{S}(f_G)/\varepsilon$ then one round of the server training loop of Algorithm (4) is (ε, δ)-DP with respect to a batch of clients [8]. Also, since a batch of clients is a random sample from database of clients[2], the privacy amplification lemma (2.1) states that a single round of Algorithm (4) is $(q\log(1 + q(e^{\varepsilon} - 1)), q\delta)$-DP. Finally, we can then use the strong composition theorem (2.6) to accumulate the DP spending over all the T rounds of Algorithm (4).

The problem, however, with this approach is that it does not provide tight guarantees of Differential Privacy. For this reason, following previous works on DP-FL ([13], [43]), we use a Moments Accountant data structure to obtain tight composition guarantees for repeated applications of a Subsampled Gaussian Mechanism. In particular, we use the Analytical Moment Accountant proposed by Yu-Xiang Wang et. al. [57] which uses Renyi differential privacy to obtain tight bounds on privacy loss under Subsampling Mechanisms.

The Analytical Moment Accountant $\mathcal{M}$ provides 2 important methods:

- $\mathcal{M}$.compose_subsampled_mechanism($\bar{\sigma}$,q):
  This method composes the privacy spending of a subsampled gaussian mechanism. The argument $\bar{\sigma}$ is the standard deviation of the Gaussian Noise normalized by the sensitivity of the gradient accumulator. Hence, since we defined $\sigma_G = \frac{zS}{qN}$ in Algorithm (4), it is evident that the normalized standard deviation we should give as input to our moments accountant is $\bar{\sigma} = \dfrac{\sigma_G}{\mathbb{S}(f_G)} = z$. Also, the parameter q given as input to the method compose_subsampled_mechanism is the sub-sampling probability of the Gaussian Mechanism, or, in our case the round participation fraction q.

- $\mathcal{M}$.get_epsilon(δ):
  This method is used to return the privacy budget spent after repeated calls to the composition method above. In particular, it takes as argument a parameter δ and finds the corresponding ε such that the overall privacy budget satisfies (ε, δ)-DP. Also, following [43], we used $\delta = \frac{1}{N^{1.1}}$ in order for the probability of the privacy leak to be less than 1 over the number of clients. The reason we chose this value for δ is that according to [25], if we set $\delta > \frac{1}{N}$, then we may permit privacy by sharing the complete records of a small number of users, which is clearly undesirable. Then,

[2]At this point, we should not that just like Augenstein et al. [13], we used a slightly different way of selecting users at each round compared to the DP-FedAvg algorithm proposed by McMahan et al. [43]. In particular, we used fixed-size federated rounds of qN clients, instead of using randomly-sized federated rounds where the users are selected independently with probability q. This has some minor effect on the overall privacy bound.

using the value $\delta = \frac{1}{N^{1.1}}$, we called $\mathcal{M}$.get_epsilon(δ) at the end of Algorithm (4) to find the overall privacy loss.

Besides measuring privacy in an actual experimental setting, the moments accountant is also useful in determining privacy in theoretical, scaled up federated scenarios. In particular, in our experiments, we used a small user population ($N = 20$) with an even smaller number of users per round ($qN = 10$). This scenario, clearly, doesn't achieve good privacy protection. However, as demonstrated in [13], the small-scale simulation scenario does indicate us the maximum noise level we can add without hurting utility. Then, while keeping noise σ_G constant, we can scale up the number of clients N and use the moments accountant to calculate the DP guarantees in the hypothetical scaled-up scenario.

The motivation behind scaling up the number of clients is that by keeping the noise σ_G constant, we can achieve better DP guarantees with more clients. In particular, our goal is to scale the number of clients such that we achieve single digit ε guarantees[3] without hurting utility. So, let's assume that in our experimental scenario we have N clients, noise scale z, noise std σ_G, round participation fraction q and $\delta = 1/N^{1.1}$, with $\sigma_G = \frac{zS}{qN}$. Under those assumptions, if we want to calculate the level of Differential Privacy in a scaled population of $N' > N$ clients, we can follow the approach used by Augenstein et al. [13]:

1. We first scale z by a factor $a = \frac{1}{z}$ so that [4] $z' = az = 1$. Then, we scale qN by a factor of a as well, so that $q'N' = aqN$. With those two scalings, we have managed to keep the actual noise σ constant, because $\sigma'_G = \frac{z'S}{q'N'} = \frac{azS}{aqN} = \frac{zS}{qN} = \sigma_G$. Hence, those scalings have **no effect on utility** of Algorithm 4.

2. Now, since $q'N' = aqN = \dfrac{qN}{z}$, we can easily calculate q' as we know N'. Hence $q' = \frac{qN}{zN'}$. Also,the privacy leak δ' for the new client population is $\delta' = 1/(N')^{1.1}$.

3. We can now use the 2 methods of the analytical moments accountant to calculate the level of differential privacy in the scaled up scenario. In particular, we can make T calls to the composition method using arguments $\bar{\sigma} = z' = 1$ and q', and we can then make a call to the second method to get our epsilon for the new privacy leak δ'.

At this point, should emphasize once again that the scaling process does not hurt utility, because it doesn't increase the actual level of noise (σ). The only trade-off is that the more we increase the number of users N, the more computationally expensive our training becomes. Nevertheless, in most cases, we have to accept this tradeoff if we want to attain good DP guarantees while maintaining utility.

FedVAESepLDP

We will now use the exact same algorithm as before, but this time, we will use Local Differential Privacy instead of Central Differential Privacy. This means that we will again use 1 VAE for every class but this time, the users are responsible for implementing their own privacy mechanisms. Hence, this leads us to the following training procedure, which is described in detail in in Algorithm (5):

1. The server initializes C VAEs, one for each class.

2. The server randomly selects qN clients out of the pool of N clients.

[3] When we say single-digit ε guarantees we are implying that $\varepsilon \simeq 1$, which is a relatively good privacy protection.

[4] The reason picked $z' = 1$ is that we scaled the level of privacy to larger populations just like in [13]. Also, as demonstrated by McMahan et al. [43], in order attain acceptable utility, we generally have to set $z' \geq 1$.

3. The server sends the current weights of the decoders of the C VAEs to the selected clients.
4. The selected clients load the weights of the decoders sent to them by the server. Then, the clients use their local samples corresponding to the i-th class to train the i-th VAE using 1 epoch of mini-batch SGD. This process is repeated for all C classes.
5. The clients calculate the weight differences Δ_i of the C decoders by subtracting the decoder weights sent by the server from the local decoder weights after training.
6. The clients clip the C weight difference vectors Δ_i so that each of those vectors has L_2 norm bounded by $S/\sqrt{C}$ and their concatenation has L_2 norm bounded by S.
7. The clients add Gaussian noise with standard deviation σ_L in order to enforce differential privacy. The choice of σ_L is made by the clients, depending on the level of privacy they want to achieve. Then, they use their local Moments Accountant in order to keep track of the local privacy loss.
8. The clients send their noisy update vectors for each one of the C decoders to the server.
9. The server aggregates the update vectors and updates the decoders of its C VAE models using the aggregated gradients.
10. The process continues from step 2 for $T-1$ more rounds.
11. **Data Generation**: For every class $i = \{1, \dots, C\}$ we can feed the decoder of the i-th VAE of the server with random Gaussian noise $\mathcal{N}(0, I)$ in order to generate data for class i.

The above training process guarantees user-level local differential privacy. In this case, however, the sensitivity changes. In fact, since the clients don't trust the server, the gradients have to be Differentially Private as soon as they leave the client. This means that in order to measure the level of Differential Privacy, we have to examine the L_2 sensitivity of the following query function[5]:

$$f_L(k) = \Delta_k \tag{4.4.7}$$

where Δ_k is the concatenation of the gradient updates of the client k. Hence, given that $\|\Delta_k\| \leq S$ due to gradient clipping, then $\mathbb{S}(f_L) = S$. As a result, the normalized standard deviation $\bar{\sigma}$ we give as input to composition method of the moments accountant is:

$$\bar{\sigma} = \frac{\sigma_L}{\mathbb{S}(f_L)} = \frac{\sigma_L}{S}$$

Also, for the second argument of the moment accountant, we use $q = 1$, as there isn't any kind of subsampling when it comes to an individual client's data: The client is viewed separately from the server and at each iteration where a client is involved, the client accesses all its data and not just a URS of that data.

By explaining how the moment accountant can be used to keep track of LDP guarantees, we have given a way for clients to measure their privacy loss across training. However, since the clients have to choose a value for σ_L prior to training, they need to approximately know what value for σ_L they have to pick in order to achieve their desired privacy budget. So, let's assume that a client k wants to achieve $(\varepsilon_k, \delta_k)$-DP guarantees,

[5]Note that the query function f_L is different than the query function f_G we used for the previous algorithm, as in this case we assume that the adversary (or the untrusted server) may have direct access to the gradient update vectors Δ_k prior to their being aggregated

while participating in 1 federated round. Then according to Theorem (2.3), the optimal Gaussian mechanism to achieve $(\varepsilon_k, \delta_k)$-DP occurs if we set:

$$\sigma_L = \frac{\left(\xi + \sqrt{\xi^2 + \varepsilon_k}\right) \cdot S}{\varepsilon_k \sqrt{2}} \tag{4.4.8}$$

where ξ can be found if we use bisection to solve the equation:

$$\operatorname{erfc}(\xi) - e^{\varepsilon_k} \operatorname{erfc}\left(\sqrt{\xi^2 + \varepsilon_k}\right) = 2\delta_k \tag{4.4.9}$$

Hence, using this method, the client can find out the level of noise they need to add so as to achieve $(\varepsilon_k, \delta_k)$-DP if they participate in 1 federated round. However, in reality, the clients don't participate in only 1 federated round. So, let's assume that the client k participates in T_k federated rounds. Hence, since the moments accountant offers $(O(\varepsilon \cdot \sqrt{T}), \delta)$-DP guarantees for T runs of an (ε, δ)-DP Gaussian mechanism (See subsection 2.2.7), then if the client uses Equation (4.4.8) to determine σ_L, the DP guarantees it will get over T_k rounds are $(O(\varepsilon_k \sqrt{T_k}), \delta_k)$. This means that if the client wants $(\varepsilon_k, \delta_k)$-DP guarantees while participating in T_k federated rounds, then the client must set:

$$\sigma_L = \frac{\left(\xi + \sqrt{\xi^2 + \varepsilon_k / \sqrt{T_k}}\right) \cdot S}{\varepsilon_k \sqrt{2 / T_k}} \tag{4.4.10}$$

and find ξ through bisection by solving:

$$\operatorname{erfc}(\xi) - e^{\varepsilon_k / \sqrt{T_k}} \operatorname{erfc}\left(\sqrt{\xi^2 + \varepsilon_k / \sqrt{T_k}}\right) = 2\delta_k \tag{4.4.11}$$

If we now view T_k as a random variable, then due to the fact that the server randomly samples clients with probability q, we expect that $E[T_k] = q \cdot T$ where T is the total number of federated rounds and q is the round participation fraction. This means that we can approximate T_k with qT and plug the approximation into equations (4.4.10) and (4.4.11). This would yield an approximate noise level σ_L the client would need to achieve $(\varepsilon_k, \delta_k)$-LDP while participating in qT training rounds. However, this is just an approximation: The exact privacy guarantees can only be calculated through the Moments Accountant present in the client. Additionally, the federated protocol can also be modified so that if a client has participated in more training rounds than what their privacy budget allows, then the they may reject further participations.

So far, we have analyzed the privacy guarantees on a client level. However, we should also examine the impact the local noise σ_L has on the server as well. For simplicity, we will assume that all clients use the same noise level σ_L. Then, if we use $\bar{\Delta}_k$ to denote the gradient updates of the k-th client before adding noise, Δ_k the gradient updates after adding noise and $X_k \sim \mathcal{N}(0, I \cdot \sigma_L^2)$ the noise added by the client, then in the server training loop we will have:

$$\begin{aligned} \Delta &= \frac{1}{qN} \sum_{k \in C^t} \Delta_k = \\ &= \frac{1}{qN} \sum_{k \in C^t} \left(\bar{\Delta}_k + X_k\right) = \\ &= \frac{1}{qN} \sum_{k \in C^t} \bar{\Delta}_k + \frac{1}{qN} \sum_{k \in C^t} X_k = \\ &= \frac{1}{qN} \sum_{k \in C^t} \bar{\Delta}_k + \frac{1}{qN} \sum_{k \in C^t} X_k \end{aligned} \tag{4.4.12}$$

The quantity $Y = \frac{1}{qN} \sum_{k \in C^t} X_k$ is a random vector, which is the average of qN i.i.d Gaussian random vectors. If we now denote the i-th component of vectors X and Y as $X^{(i)}$ and $Y^{(i)}$ respectively, then $Y^{(i)}$ is the weighted sum of qN i.i.d Gaussian random variables. Hence, $Y^{(i)}$ is also Gaussian with:

$$E[Y^{(i)}] = E\Big[\frac{1}{qN} \sum_{k \in C^t} X_k^{(i)}\Big] = \frac{1}{qN} \sum_{k \in C^t} E[X_k^{(i)}] = 0$$

and:

$$\begin{aligned} \mathrm{Var}(Y^{(i)}) &= \mathrm{Var}\Big(\frac{1}{qN} \sum_{k \in C^t} X_k^{(i)}\Big) = \\ &= \frac{1}{q^2N^2} \sum_{k \in C^t} \mathrm{Var}(X_k^{(i)}) = \\ &= \frac{1}{q^2N^2} \sum_{k \in C^t} \sigma_L^2 = \\ &= \frac{1}{q^2N^2} qN\sigma_L^2 = \\ &= \frac{\sigma_L^2}{qN} \end{aligned}$$

Hence, $Y \sim \mathcal{N}(0, I\frac{\sigma_L^2}{qN})$. This leads us to the following theorem:

Theorem 4.1 *Using global Gaussian noise with std σ_G in the CDP setting of Algorithm (4) is exactly equivalent (noise-wise) to using local Gaussian noise with std $\sigma_L = \sigma_G\sqrt{qN}$ in the LDP setting of Algorithm (5).*

This theorem is very important as it will later help us compare the privacy guarantees of CDP and LDP settings.

FedVAEUniCDP

In this federation scheme, we will use 1 VAE to generate data for all classes, while utilizing Central Differential Privacy. In order to achieve that, we have make a slight variation to the decoder of a Variational Autoencoder. In a conventional VAE, the encoder takes as input a sample $\mathbf{x}$ and generates 2 vectors $\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x \in \mathbb{R}^{n_U}$. Then we generate a random noise vector $\mathbf{u} \sim \mathcal{N}(\boldsymbol{\mu}_x, \mathrm{diag}(\boldsymbol{\sigma}_x)^2)$ and we give this vector as input to the decoder. This means that the decoder has n_U inputs. However, since we also want to choose the class our decoder generates samples from, we will need to condition the decoder on the class label y. For this reason, we will slightly alter the decoder so that it has $n_U + C$ inputs instead of n_U. Then, whenever we want our decoder to generate samples with label y, we will provide the encoder with a vector $\mathbf{v}$ which is the concatenation of a noise vector $\mathbf{z}$ and the one-hot encoding of the label y. This idea of a conditional decoder is very similar to the one proposed in [50] and allows us to use just 1 decoder so as to generate samples from whichever class we want. By making this modification in the FedVAESepCDP scenario, the training process becomes:

1. The server initializes 1 VAEs for all classes.
2. The server randomly selects qN clients out of the pool of N clients.
3. The server sends the current weights of the decoder of its VAE to the selected clients.

4. The selected clients load the weights of the decoder sent to them by the server. Then, the clients use their samples to train the local VAE using 1 epoch of mini-batch SGD. During training, every local sample $\mathbf{x}$ is passed through the encoder to generate the mean and std vectors $\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x$. Then, a noise vector $\mathbf{u} \sim \mathcal{N}(\boldsymbol{\mu}_x, \mathrm{diag}(\boldsymbol{\sigma}_x)^2)$ is generated, this vector becomes concatenated with the one-hot encoding of the label of $\mathbf{x}$, and finally the concatenated vector is then passed through the decoder.

5. The clients calculate the weight difference Δ of their decoder by subtracting the decoder weights sent by the server from the local decoder weights after training.

6. The clients clip the weight difference vector Δ so that it has L_2 norm bounded by S.

7. The clients send their clipped non-noisy update vector Δ to the server.

8. The server aggregates the update vectors for the decoder and adds Gaussian noise of std σ_G in order to ensure Central Differential Privacy.

9. The server updates its decoder using the aggregated gradients.

10. The process continues from step 2 for $T-1$ more rounds.

11. **Data Generation**: In order to generate samples of the i-th class, we have to generate a random Gaussian vector $\mathbf{u} \sim \mathcal{N}(0, I)$ with dimension n_U, concatenate that vector with the one-hot encoding of the desired label i and then feed the result into the decoder.

This process is described in detail in Algorithm (6) and is very similar to Algorithm (4), except that in this case, we use 1 VAE for all classes and not 1 VAE for each class. In fact, using 1 VAE for all classes has many advantages, such as:

- **Reduced communication overhead**: In the FedVAEUniCDP scenario, the clients only need to send to the server the weight updates of 1 decoder instead of C decoders. This significantly reduces the amount of information the clients need to send, almost by a factor of $1/C$.

- **Reduced computational cost**: Since the training process of FedVAEUniCDP involves just 1 VAE instead of C VAEs, the computational complexity of training is much better not just for the clients, but also for the server.

- **Rich Latent Space**: in the FedVAEUniCDP scenario, the VAE that is responsible for generating all classes, has a much richer latent space than a VAE which generates a single class. This opens the possibility for more meaningful latent representations of the input samples, which may be utilized to gain insights into the training set.

The DP guarantees of the FedVAEUniCDP scenario can be calculated exactly in the same way as we did with FedVAESepCDP. The reason for this is that since we want to enforce Central Differential Privacy, the gradient aggregator is the same as in (4.4.4), and thus, the sensitivity of this aggregator is the same as in (4.4.6) (i.e. $\mathbb{S}(f) = S$).

FedVAEUniLDP

In this scenario, we will again use 1 VAE for all classes, but this time we want to enforce Local Differential Privacy instead of Central Differential Privacy. This leads us to the following training process:

1. The server initializes 1 VAEs for all classes.

2. The server randomly selects qN clients out of the pool of N clients.
3. The server sends the current weights of the decoder of its VAE to the selected clients.
4. The selected clients load the weights of the decoder sent to them by the server. Then, the clients use their samples to train the local VAE using 1 epoch of mini-batch SGD. During training, every local sample $\mathbf{x}$ is passed through the encoder to generate the mean and std vectors $\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x$. Then, a noise vector $\mathbf{u} \sim \mathcal{N}(\boldsymbol{\mu}_x, \text{diag}(\boldsymbol{\sigma}_x)^2)$ is generated, this vector becomes concatenated with the one-hot encoding of the label of $\mathbf{x}$, and finally the concatenated vector is then passed through the decoder.
5. The clients calculate the weight difference Δ of their decoder by subtracting the decoder weights sent by the server from the local decoder weights after training.
6. The clients clip the weight difference vector Δ so that it has L_2 norm bounded by S.
7. The clients add Gaussian noise with standard deviation σ_L in order to enforce local differential privacy. The choice of σ_L is made by the clients, depending on the level of privacy they want to achieve. Then, the clients can use their local Moments Accountant to calculate the privacy loss.
8. The clients send their noisy update vectors Δ to the server.
9. The server aggregates the update vectors.
10. The server updates its decoder using the aggregated gradients.
11. The process continues from step 2 for $T-1$ more rounds.
12. **Data Generation**: In order to generate samples of the i-th class, we have to generate a random Gaussian vector $\mathbf{u} \sim \mathcal{N}(0, I)$ with dimension n_U, concatenate that vector with the one-hot encoding of the desired label i and then feed the result into the decoder.

This process is described in detail in Algorithm (7). Additionally, the LDP guarantees in this scenario are exactly the same as in the FedVAESepLDP case, as the update vectors Δ have the same bound to their L_2 norm (prior to adding noise).

Algorithm 4: Federated separate VAE with central DP (FedVAESepCDP)

```
parameters: round participation fraction $q \in (0, q]$, requested DP privacy leak $\delta$,
  total number of users $N \in \mathbb{N}$, total number of rounds $T \in \mathbb{N}$, dp noise scale
  $z \in \mathbb{R}^+$, gradient clipping parameter $S \in \mathbb{R}^+$, params of decoders $\Theta$

Initialize encoder $\Phi^0$, decoder $\Theta^0$ and DP accountant $\mathcal{M}$

$\sigma_G \leftarrow \frac{zS}{qN}$
foreach round $t \in \{1, 2, \dots, T\}$ do
    $C^t \leftarrow$ (sample of qN distinct users)
    foreach client $k \in C^t$ do
        $\Delta_k^{t+1} \leftarrow$ ClientTrain$(k, \Theta)$
    end
    $\Delta^{t+1} = \frac{1}{qN} \sum_{k \in C^t} \Delta_k^{t+1}$
    $\Theta^{t+1} \leftarrow \Theta^t + \Delta^{t+1} + \mathcal{N}(0, I\sigma_G^2)$
    $\mathcal{M}$.compose_subsampled_mechanism$(z, q)$
end
print $\mathcal{M}$.get_epsilon$(\delta)$

Function ClientTrain$(k, \Theta^0)$:
    parameters: Batch size $B \in \mathbb{N}$, learning rate $\eta \in \mathbb{R}^+$, client id $k$, gradient
      clipping term $S \in \mathbb{R}^+$, Size of latent dim $n_U$, number of classification
      classes $C \in \mathbb{N}$, local encoder weights for the $i$-th class $\Phi_i$, local decoder
      weights for the $i$-th class $\Theta_i$, encoder $E(V; \Phi_i)$, decoder $D(U; \Theta_i)$, loss
      function $l(x, \hat{x}; \Phi_i, \Theta_i)$, server decoder weights for the $i$-th class $\Theta_i^0$

    foreach $i \in \{1, \dots, C\}$ do
        $\Theta_i \leftarrow \Theta_i^0$
        $\mathcal{B}_i \leftarrow$ (split data of the $i-$th class of the $k-$th client into batches of size $B$)
        foreach $b \in \mathcal{B}_i$ do
            $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow E(b; \Phi_i)$
            $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow$ flatten$(\boldsymbol{\mu})$, flatten$(\boldsymbol{\sigma})$
            $U \leftarrow$ (Generate 1 sample vector from $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})^2)$)
            $U \leftarrow$ (Reshape U to size $B \times n_U$)
            $\hat{b} \leftarrow D(U; \Theta_i)$
            $(\Phi_i, \Theta_i) \leftarrow (\Phi_i, \Theta_i) - \eta \nabla l(b, \hat{b}; \Phi_i, \Theta_i)$
        end
        $\Delta_i = \Theta_i - \Theta_i^0$ //Calculate weight differences after train
        $\Delta_i = \Delta_i \cdot \min\left(1, \frac{S}{\sqrt{C}\|\Delta_i\|}\right)$ //Gradient Clipping
    end
    $\Delta \leftarrow$ concat$(\Delta_1, \Delta_2, \dots, \Delta_C)$ //Due to clipping, $\|\Delta\|_2 \leq S$
    return $\Delta$
End Function
```

Algorithm 5: Federated seperate VAE with local DP (FedVAESepLDP)

```
parameters: round participation fraction $q \in (0, q]$, requested DP privacy leak $\delta$,
  total number of users $N \in \mathbb{N}$, total number of rounds $T \in \mathbb{N}$, dp noise scale
  $z \in \mathbb{R}^+$, gradient clipping parameter $S \in \mathbb{R}^+$

Initialize encoder $\Phi^0$, decoder $\Theta^0$
Initialize dp accountant for each client $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_N\}$
foreach round $t \in \{1, 2, \ldots, T\}$ do
    $C^t \longleftarrow$ (sample of qN distinct users)
    foreach client $k \in C^t$ do
        $\Delta_k^{t+1} \longleftarrow \text{ClientTrain}(k, \Theta^0)$
    end
    $\Delta^{t+1} = \frac{1}{qN} \sum_{k \in C^t} \Delta_k^{t+1}$
    $\Theta_D^{t+1} \longleftarrow \Theta_D^t + \Delta^{t+1}$
end
print $\mathcal{M}$.get_epsilon($\delta$)

Function ClientTrain($k, \Theta^0, \mathcal{M}_k$):
    parameters: Batch size $B \in \mathbb{N}$, learning rate $\eta \in \mathbb{R}^+$, client id $k$, gradient
      clipping term $S \in \mathbb{R}^+$, Size of latent dim $n_U$, number of classification
      classes $C \in \mathbb{N}$, local encoder weights for the $i$-th class $\Phi_i$, local decoder
      weights for the $i$-th class $\Theta_i$, encoder $E(V; \Phi_i)$, decoder $D(U; \Theta_i)$, loss
      function $l(x, \hat{x}; \Phi_i, \Theta_i)$, server decoder weights for the $i$-th class $\Theta_i^0$, local
      noise standard deviation $\sigma_L$

    foreach $i \in \{1, \ldots, C\}$ do
        $\Theta_i \longleftarrow \Theta_i^0$
        $\mathcal{B}_i \longleftarrow$ (split data of the $i-$th class of the $k-$th client into batches of size $B$)
        foreach $b \in \mathcal{B}_i$ do
            $\boldsymbol{\mu}, \boldsymbol{\sigma} \longleftarrow E(b; \Phi_i)$
            $\boldsymbol{\mu}, \boldsymbol{\sigma} \longleftarrow \text{flatten}(\boldsymbol{\mu}), \text{flatten}(\boldsymbol{\sigma})$
            $U \longleftarrow$ (Generate 1 sample vector from $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})^2)$
            $U \longleftarrow$ (Reshape U to size $B \times n_U$)
            $\hat{b} \longleftarrow D(U; \Theta_i)$
            $(\Phi_i, \Theta_i) \longleftarrow (\Phi_i, \Theta_i) - \eta \nabla l(b, \hat{b}; \Phi_i, \Theta_i)$
        end
        $\Delta_i = \Theta_i - \Theta_i^0$ //Calculate weight differences after train
        $\Delta_i = \Delta_i \cdot \min\left(1, \frac{S}{\sqrt{C}\|\Delta_i\|}\right)$ //Gradient Clipping
    end
    $\Delta \longleftarrow \text{concat}(\Delta_1, \Delta_2, \ldots, \Delta_C)$ //Due to clipping, $\|\Delta\|_2 \leq S$
    $\Delta \longleftarrow \Delta + \mathcal{N}(0, I \cdot \sigma_L^2)$ //Add noise
    $\mathcal{M}_k$.compose_subsampled_mechanism($\sigma_L / S, 1$)
    return $\Delta$

End Function
```

Algorithm 6: Federated unified VAE with global DP (FedVAEUniCDP)

```
parameters: round participation fraction $q \in (0, q]$, requested DP privacy leak $\delta$,
  total number of users $N \in \mathbb{N}$, total number of rounds $T \in \mathbb{N}$, dp noise scale
  $z \in \mathbb{R}^+$, gradient clipping parameter $S \in \mathbb{R}^+$

Initialize encoder $\Phi^0$, decoder $\Theta^0$, DP accountant $\mathcal{M}$

$\sigma_G \leftarrow \frac{zS}{qN}$
foreach round $t \in \{1, 2, \ldots, T\}$ do
    $C^t \leftarrow$ (sample of qN distinct users)
    foreach client $k \in C^t$ do
        $\Delta_k^{t+1} \leftarrow \text{ClientTrain}(k, \Theta^0)$
    end
    $\Delta^{t+1} = \frac{1}{qN} \sum_{k \in C^t} \Delta_k^{t+1}$
    $\Theta_D^{t+1} \leftarrow \Theta_D^t + \Delta^{t+1} + \mathcal{N}(0, I\sigma_G^2)$
    $\mathcal{M}$.compose_subsampled_mechanism($z, q$)
end
print $\mathcal{M}$.get_epsilon($\delta$)

Function ClientTrain($k, \Theta^0$):
    parameters: Batch size $B \in \mathbb{N}$, learning rate $\eta \in \mathbb{R}^+$, client id $k$, gradient
      clipping term $S \in \mathbb{R}^+$, Size of latent dim $n_U$, number of classification
      classes $C \in \mathbb{N}$, local encoder weights $\Phi$, local decoder weights $\Theta$, encoder
      $E(V; \Phi)$, decoder $D(U; \Theta)$, loss function $l(x, \hat{x}; \Phi, \Theta)$, server decoder
      weights $\Theta^0$

    $\Theta \leftarrow \Theta^0$
    $\mathcal{B} \leftarrow$ (split data of the $k-$th client into batches of size $B$)
    foreach $(\mathbf{X}, \mathbf{y}) \in \mathcal{B}$ do
        $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow E(\mathbf{X}; \Phi)$
        $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow \text{flatten}(\boldsymbol{\mu}), \text{flatten}(\boldsymbol{\sigma})$
        $U \leftarrow$ (Generate 1 sample vector from $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma})^2)$)
        $U \leftarrow$ (Reshape U to size $B \times n_U$)
        $U \leftarrow [U, \text{one\_hot}(\mathbf{y})]$//Concatenate $U$ with one hot encoding of labels
        $\hat{b} \leftarrow D(U; \Theta)$
        $(\Phi, \Theta) \leftarrow (\Phi, \Theta) - \eta \nabla l(b, \hat{b}; \Phi, \Theta)$
    end
    $\Delta = \Theta - \Theta^0$//Calculate weight differences after train
    $\Delta = \Delta \cdot \min\left(1, \frac{S}{\|\Delta\|}\right)$//Gradient Clipping
    return $\Delta$

End Function
```

Algorithm 7: Federated unified VAE with local DP (FedVAEUniLDP)

```
parameters: round participation fraction $q \in (0, q]$, requested DP privacy leak $\delta$,
  total number of users $N \in \mathbb{N}$, total number of rounds $T \in \mathbb{N}$, dp noise scale
  $z \in \mathbb{R}^+$, gradient clipping parameter $S \in \mathbb{R}^+$

Initialize encoder $\Phi^0$, decoder $\Theta^0$, DP accountant
Initialize dp accountant $\mathcal{M}$

$\sigma \leftarrow \frac{zS}{qN}$
foreach round $t \in \{1, 2, \ldots, T\}$ do
    $C^t \leftarrow$ (sample of qN distinct users)
    foreach client $k \in C^t$ do
        $\Delta_k^{t+1} \leftarrow$ ClientTrain($k, \Theta^0, \mathcal{M}_k$)
    end
    $\Delta^{t+1} = \frac{1}{qN} \sum_{k \in C^t} \Delta_k^{t+1}$
    $\Theta_D^{t+1} \leftarrow \Theta_D^t + \Delta^{t+1}$
end

Function ClientTrain($k, \Theta^0$):
    parameters: Batch size $B \in \mathbb{N}$, learning rate $\eta \in \mathbb{R}^+$, client id $k$, gradient
      clipping term $S \in \mathbb{R}^+$, Size of latent dim $n_U$, number of classification
      classes $C \in \mathbb{N}$, local encoder weights $\Phi$, local decoder weights $\Theta$, encoder
      $E(V; \Phi)$, decoder $D(U; \Theta)$, loss function $l(x, \hat{x}; \Phi, \Theta)$, server decoder
      weights $\Theta^0$, local noise standard deviation $\sigma_L$

    $\Theta \leftarrow \Theta^0$
    $\mathcal{B} \leftarrow$ (split data of the $k-$th client into batches of size $B$)
    foreach $(\mathbf{X}, \mathbf{y}) \in \mathcal{B}$ do
        $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow E(\mathbf{X}; \Theta)$
        $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow$ flatten($\boldsymbol{\mu}$), flatten($\boldsymbol{\sigma}$)
        $U \leftarrow$ (Generate 1 sample vector from $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})^2)$)
        $U \leftarrow$ (Reshape U to size $B \times n_U$)
        $U \leftarrow [U, \text{one\_hot}(\mathbf{y})]$//Concatenate $U$ with one hot encoding of labels
        $\hat{b} \leftarrow D(U; \Theta)$
        $(\Phi, \Theta) \leftarrow (\Phi, \Theta) - \eta \nabla l(b, \hat{b}; \Phi, \Theta)$
    end
    $\Delta = \Theta - \Theta^0$//Calculate weight differences after train
    $\Delta = \Delta \cdot \min\left(1, \frac{S}{\|\Delta\|}\right)$//Gradient Clipping
    $\Delta \leftarrow \Delta + \mathcal{N}(0, I\sigma_L^2)$
    $\mathcal{M}$.compose_subsampled_mechanism($\sigma_L / S, q$)
    return concat($\Delta_1, \Delta_2, \ldots, \Delta_C$);
End Function
```

4.4.3 Implementation

Let's now discuss some aspects of the implementation of Federated VAEs. As we previously discussed, we have used 2 different datasets in our experiments: The EMNIST and the Epilepsy dataset. Hence, for each one of those datasets, we have used a different Neural Network architecture for our VAEs. In particular, when it comes to EMNIST, we used a Convolutional Variational Autoencoder architecture (CVAE), where the **encoder** is:

- Input layer of size 28×28
- Conv2D layer with 32 filters and stride 2
- 2×Conv2D layers with 64 filters and stride 2
- Flatten layer
- Dense layer with $2 \cdot$ LATENT_DIM nodes

and the **decoder** is:

- Input layer of size LATENT_DIM + C
- Dense layer with 1568 units
- Reshape layer with target shape $7 \times 7 \times 32$
- Inverse Conv2D layer with 32 filters and stride 1
- Inverse Conv2D layer with 64 filters and stride 1
- Inverse Conv2D layer with 64 filters and stride 2
- Inverse Conv2D layers with 32 filters and stride 2
- Inverse Conv2D layers with 1 filter and stride 1

In the figure below, we can see a visualization of this variational autoencoder for the EMNIST dataset:

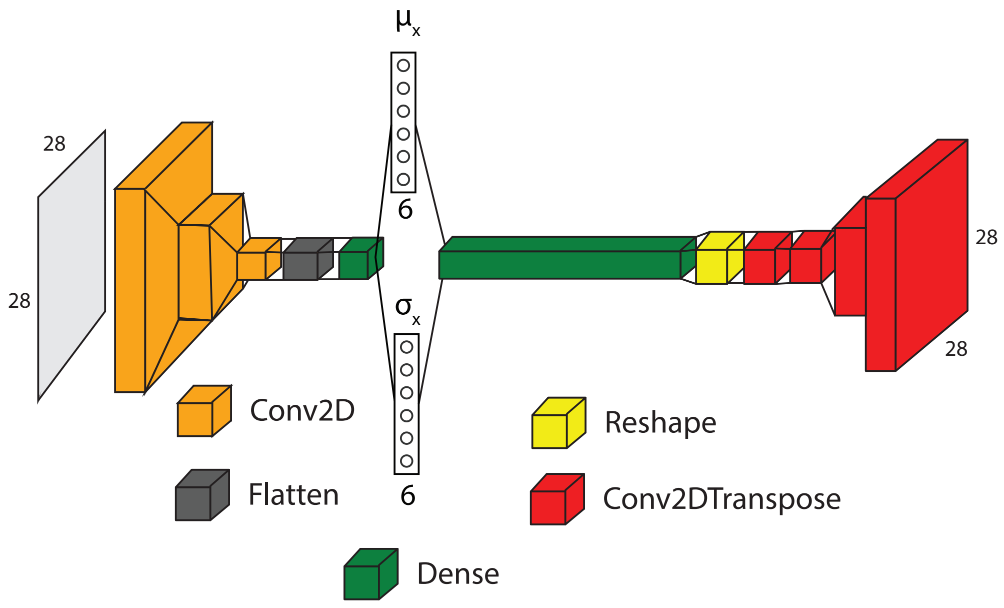


Figure 4.4.3: Visualization of the EMNIST Variational Autoencoder

The reason we chose this architecture for the EMNIST VAE is that it is very similar to the architecture used in an official tensorflow tutorial for Convolutional VAEs [14]. In this tutorial, this convolutional architecture for VAEs seems to demonstrate a good generative performance in the centralized MNIST dataset, and that's why we decided to adapt this architecture to our setting.

When it comes to the Epilepsy dataset, we used a standard dense architecture with relu activations. In particular, the architecture of our **encoder** was:

- Input layer of size 178
- Dense RELU layer with 128 units
- Dense RELU layer with 64 units
- Dense RELU layer with 32 units
- Dense RELU layer with 16 units
- Dense layer with $2 \cdot$ LATENT_DIM nodes

and the architecture of our **decoder** was:

- Input layer of size LATENT_DIM
- Dense RELU layer with 16 units
- Dense RELU layer with 32 units
- Dense RELU layer with 64 units
- Dense RELU layer with 128 units
- Dense layer with 178 units (no activation)

The architecture of the VAE for the Epilepsy dataset was partially inspired by the architecture used for Tabular VAEs by the Synthetic Data Vault project of MIT [55]. A schematic diagram of our Epilepsy VAE architecture can be shown in figure (4.4.4).

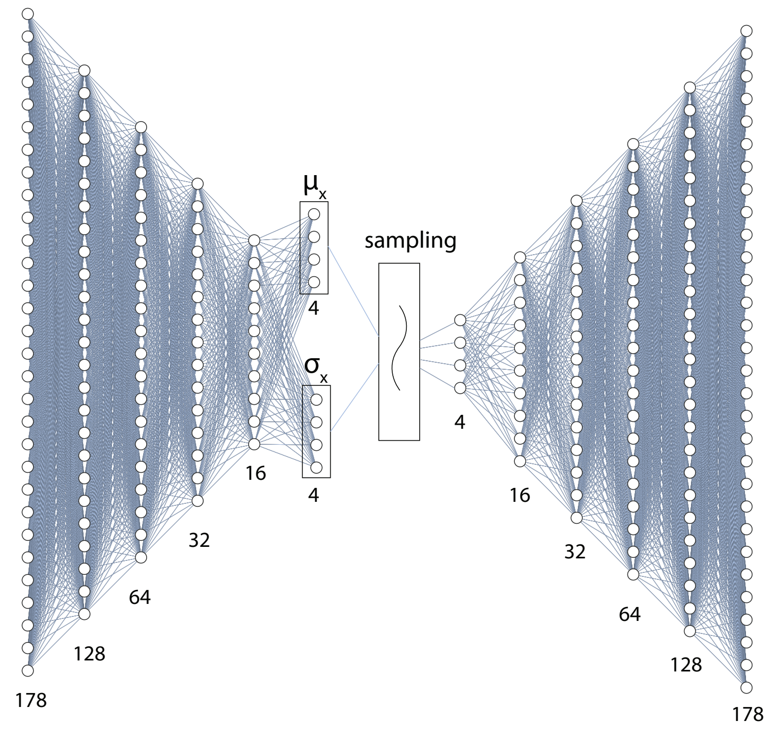


Figure 4.4.4: Visualization of the Epilepsy Variational Autoencoder

Evaluation

Another important aspect of the implementation of Federated VAEs is finding a way to assess the quality of the artificial data generated by the VAEs. Hence, although many metrics have been proposed for this task, we will use the accuracy of the student network for our experiments.

The student network is a classification network that will be trained using the artificial data generated from our VAEs. Then, we will evaluate the accuracy of the student network on the real test set. Hence, if our VAEs generate a high quality artificial dataset that is very similar to the real training set, then we expect the accuracy of the student network to be high. By the same token, if the quality of the data generated is low, we expect to have lower accuracy on the test set. Hence, we can use the accuracy of the student network as a metric for the quality of the generated data.

However, although accuracy is generally a good metric for balanced classification tasks, it is not that good when it comes to imbalanced classification. For this reason, as a metric for the Epilepsy dataset, we used f1-score instead of accuracy, due to the fact that the Epilepsy dataset is highly imbalanced, with 80% of samples corresponding to label 0 and only 20% of the samples corresponding to label 1. On the contrary, when it comes to the EMNIST dataset, we used accuracy instead of f1-score, due to the fact that the dataset was balanced.

For clarification purposes, we present the formulas used to calculate Accuracy and F1-score for a classification task. Note that the formula for accuracy works for both binary and multi-label classification problems, whereas the formula for F1-score works only for binary classification tasks:

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}}$$

and:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

where:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive+False Positive}}$$

and:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive+False Negative}}$$

When it comes to the architecture of the Student Network, we used a Convolutional Neural Network for EMNIST and a standard multi-layer neural network for Epilepsy. In particular, the architecture we used for the EMNIST dataset is the following:

- Input layer of size 28×28
- Conv2D layer with 32 filters, 3×3 kernel and stride 1
- Max Pooling 2D layer with pool size 2×2
- Flatten layer
- Dense layer with 100 units
- Dense softmax layer with $C = 10$ units

This architecture was inspired by online tutorials on designing high accuracy CNN classifiers for the MNIST dataset.

On the other hand, the architecture of the student network we used for the Epilepsy dataset is the following:

- Input layer of size 178
- Dense RELU layer with 1024 units
- Dropout layer with dropout rate 0.25
- Flatten layer
- Dense sigmoid layer with 1024 units
- Dropout layer with dropout rate 0.4
- Dense sigmoid layer with 2 units

This architecture was chosen after a lot of trial, error and experimentation, where we found that the architecture performs relatively well in classifying samples of the epilepsy dataset.

4.4.4 Challenges

During the implementation of Federated VAEs, we faced several challenges. The most important one was that Federated VAEs exhibited a very unstable behavior during training, which in many instances caused a collapse of the student network accuracy. However, after a lot of experimentation, we found that the root cause of this behavior was an exploding loss function which caused an explosion of the gradients. Hence, in order to alleviate this problem, we introduced some level of value clipping into the loss function, so that it never exceeds the maximum limit for float32 values.

At the same time, VAEs also required a considerable level of hyperparameter tuning. In particular, we had to experiment a lot with different batch sizes, different size of latent dimension,different learning rates and different Neural Network architectures. Hence, after a lot of fine-tuning and experimentation, we managed to achieve descent performance using our federated VAEs.

4.5 Results

In this section, we are going to present the experimental results from training our federated VAEs under different noise regimes, data distribution policies and model sizes. We are also going to compare Federated DP VAEs against other models that attempt to solve the same problem as the one we are solving, such as PrivBayes and Federated DP GANs.

Before we begin, we should note that the implementation of all models and experiments was done in Python 3.7 using numpy and Tensorflow 2. Then, the experiments were run in a desktop computer with 16GB of ram, Quad-core 4GHz processor (i7-6700k) and a 6-GB Nvidia GPU card (GTX 1060 GB). Also, in order for tensorflow to be able to run on the GPU, we used Cuda and CudNN libraries, which significantly accelerated the training process.

In order to log our results during training we used Tensorboard. Then, we wrote a python script which reads Tensorboard logs, creates the experimental figures and calculates the level of Differential Privacy for every experimental setting.

4.5.1 Federated vs Centralized DP VAEs

Firstly, we are going to study the performance of Federated DP VAEs under different noise regimes. The performance evaluation will be done using the accuracy of the student network for the EMNIST dataset and using f1 score of the student network for the Epilepsy dataset. Those scores will be calculated by training the Student network on the artificial data generated from our VAEs and then evaluating the student network on the real test set.

In order to facilitate comparison, we will also use 2 different baselines:

a) The accuracy/f1 score of a Student network that is trained directly on real training data. Those real training data are the result of merging the federated datasets into a common, Centralized dataset. In this scenario, VAEs are not involved at all, but we use this metric as a baseline to measure the maximum accuracy/f1 score that our VAEs could theoretically achieve. Clearly, however, VAEs cannot compete against this baseline, as the artificial dataset generated from the VAEs will always be of lower quality than the real dataset.

b) The accuracy/f1 score that we would have if we accumulated all the data in the server, trained a centralized VAE using that data and then evaluated the performance of the student network the artificial data generated from that VAE. This baseline acts as an indicator of how much utility we are sacrificing by moving from a centralized to a federated setting.

At this stage, it should be noted that the data distribution among clients follows a uniform distribution, but in the next section we are also going to study other data distributions as well. In any case, the hyperparameters used for the EMNIST and the Epilepsy datasets are shown on Table (4.5.1) and (4.5.2) respectively.

learning rate η	latent dim n_U	batch size B	L2 clip parameter S
$2 \cdot 10^{-3}$	8	16	0.1
total users N	users per round qN	federated rounds T	noise scale z
20	10	1500	$0,\ 0.05,\ 0.1$
data distribution	seperation		
UNIFORM	1 VAE for all classes		

Table 4.5.1: Hyperparameters for EMNIST Federated VAEs

learning rate η	latent dim n_U	batch size B	L2 clip parameter S
$1 \cdot 10^{-5}$	8	16	0.1
total users N	users per round qN	federated rounds T	noise scale z
20	10	1200	$0,\ 0.1,\ 0.2,\ 0.5$
data distribution	seperation		
UNIFORM	1 VAE per class		

Table 4.5.2: Hyperparameters for Epilepsy Federated VAEs

Using those hyperparameters, we ran FedVAEUniCDP algorithm on EMNIST and FedVAESepCDP algorithm on epilepsy[6], as shown in Figures (4.5.1) and (4.5.2) respectively. In these figures, we have plotted the accuracy/f1 score of the student network when trained on data from centralized and federated VAEs under different noise regimes. Also, for more robust results, the accuracy/f1 score is calculated as the average accuracy/f1 score in 10 different runs of the experiment. Last but not least, for every federated scenario presented in Figures (4.5.1) and (4.5.2), we have calculated the respective privacy budgets in Tables (4.5.3) and (4.5.4), not just for the simulation scenarios where $N = 20$, but also for realistic, scaled-up scenarios where $N \simeq 10^4$.

[6]The reason we use FedVAESepCDP algorithm instead of FedVAEUniCDP for Epilepsy is that the Epilepsy dataset is highly imbalanced. Hence, if we used 1 VAE for all classes (i.e. use algorithm FedVAEUniCDP), then the VAE would not be efficient in generating the minority class, as the majority class would dominate the training. Hence, that's why we use FedVAESepCDP in Epilepsy, as we want the majority and the minority class to be associated with different VAEs.

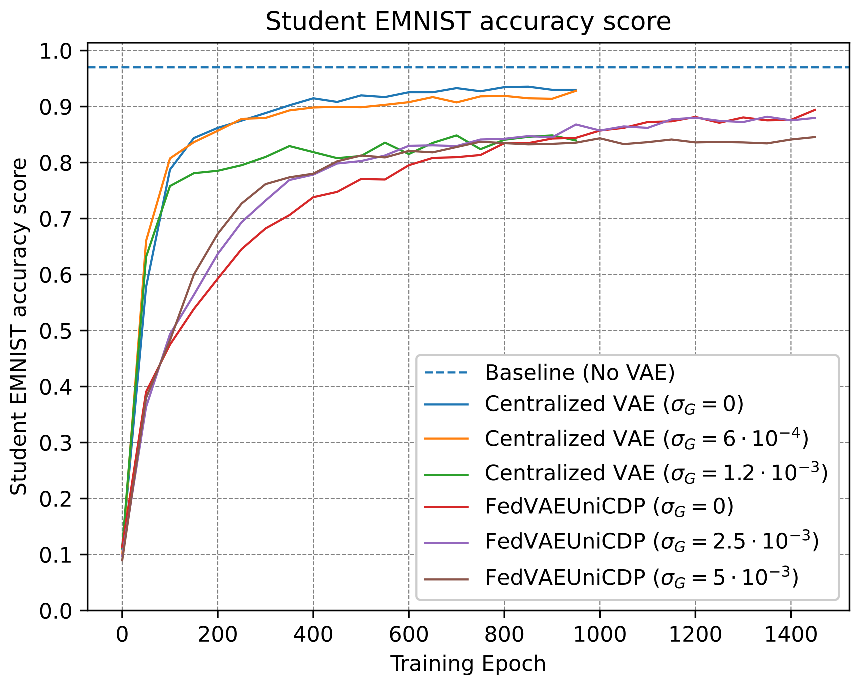


Figure 4.5.1: Accuracy of DpFedVAEs on EMNIST under different noise regimes

Model	DP in simulation (N=20)	DP in scaled population (N=30000)
Centralized VAE ($\sigma_G = 0$)	$(\infty, 1.00 \cdot 10^{-6})$	-
Centralized VAE ($\sigma_G = 6 \cdot 10^{-4}$)	$(1.26 \cdot 10^{6}, 1.00 \cdot 10^{-6})$	-
Centralized VAE ($\sigma_G = 1.2 \cdot 10^{-3}$)	$(3.17 \cdot 10^{5}, 1.00 \cdot 10^{-6})$	-
FedVAEUniCDP ($\sigma_G = 0$)	$(\infty, 3.71 \cdot 10^{-2})$	$(\infty, 1.19 \cdot 10^{-5})$
FedVAEUniCDP ($\sigma_G = 2.5 \cdot 10^{-3}$)	$(5.99 \cdot 10^{5}, 3.71 \cdot 10^{-2})$	$(2.63 \cdot 10^{0}, 1.19 \cdot 10^{-5})$
FedVAEUniCDP ($\sigma_G = 5 \cdot 10^{-3}$)	$(1.49 \cdot 10^{5}, 3.71 \cdot 10^{-2})$	$(1.28 \cdot 10^{0}, 1.19 \cdot 10^{-5})$

Table 4.5.3: (ε, δ)-DP in different scenarios of EMNIST

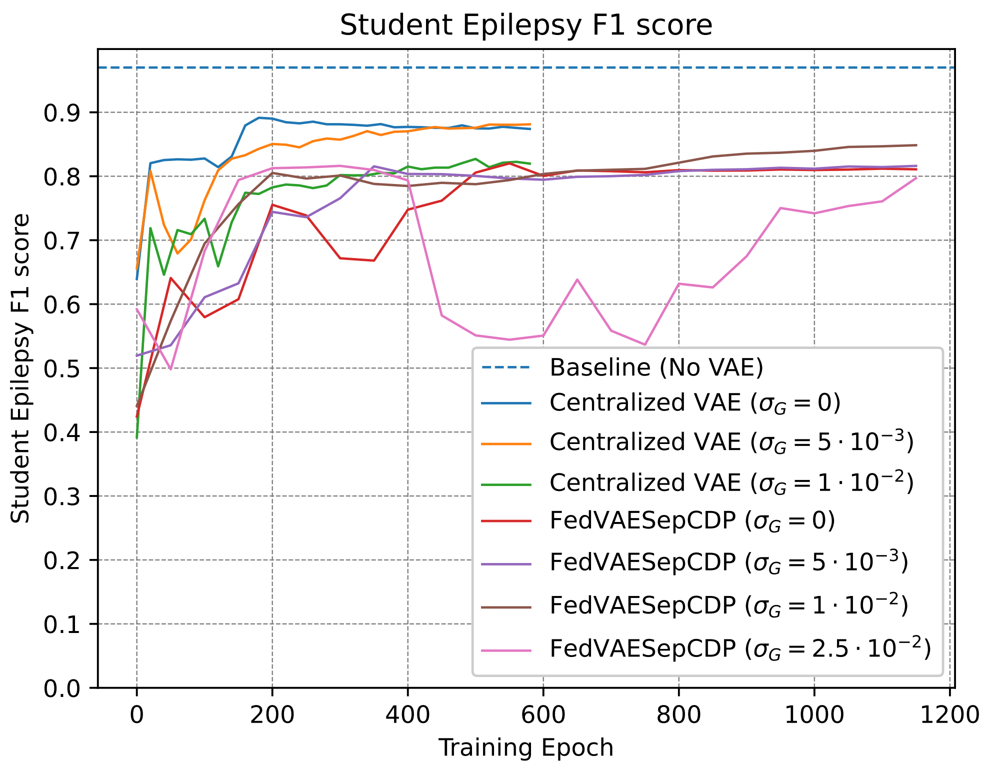


Figure 4.5.2: Accuracy of DpFedVAEs on Epilepsy under different noise regimes

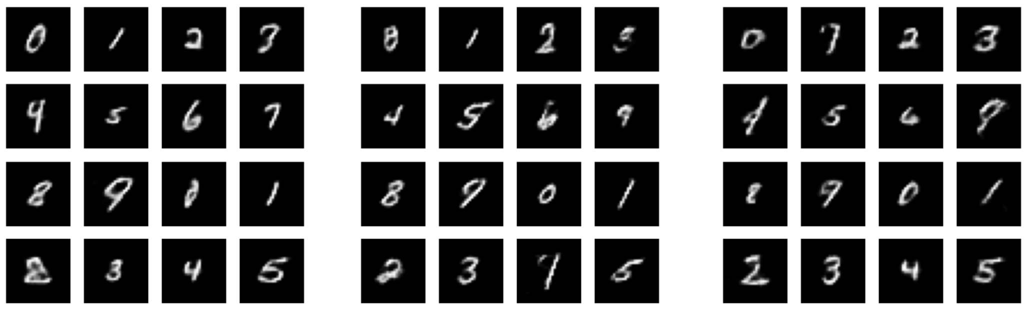
Model	DP in simulation (N=20)	DP in scaled population (N=10000)
Centralized VAE ($\sigma_G = 0$)	$(\infty, 1.00 \cdot 10^{-6})$	-
Centralized VAE ($\sigma_G = 5 \cdot 10^{-3}$)	$(1.46 \cdot 10^{5}, 1.00 \cdot 10^{-6})$	-
Centralized VAE ($\sigma_G = 1 \cdot 10^{-2}$)	$(4.29 \cdot 10^{4}, 1.00 \cdot 10^{-6})$	-
FedVAESepCDP ($\sigma_G = 0$)	$(\infty, 3.71 \cdot 10^{-2})$	$(\infty, 3.98 \cdot 10^{-5})$
FedVAESepCDP ($\sigma_G = 5 \cdot 10^{-3}$)	$(1.19 \cdot 10^{5}, 3.71 \cdot 10^{-2})$	$(3.41 \cdot 10^{0}, 3.98 \cdot 10^{-5})$
FedVAESepCDP ($\sigma_G = 1 \cdot 10^{-2}$)	$(2.92 \cdot 10^{4}, 3.71 \cdot 10^{-2})$	$(1.65 \cdot 10^{0}, 3.98 \cdot 10^{-5})$
FedVAESepCDP ($\sigma_G = 2.5 \cdot 10^{-2}$)	$(4.01 \cdot 10^{3}, 3.71 \cdot 10^{-2})$	$(6.44 \cdot 10^{-1}, 3.98 \cdot 10^{-5})$

Table 4.5.4: (ε, δ)-DP in different scenarios of Epilepsy

By examining Figures (4.5.1), (4.5.2) and tables (4.5.3), (4.5.4), we can draw the following conclusion for our Federated VAEs:

When VAEs are trained on EMNIST without the presence of noise, the Federated VAE model (i.e. FedVAEUniCDP) seems to have very similar accuracy to the Centralized VAE model, with a difference of only $2-3$ percentage points. If we then add a moderate amount of noise ($\sigma_G = 2.5 \cdot 10^{-3}$) to the federated model, then the drop in accuracy is rather negligible, but the gain in privacy is significant: From having no privacy, we end up having $\varepsilon \simeq 10^5$ CDP privacy budget in simulation and single-digit ε CDP in a scaled population of 30000 clients. Then, when we double the level of noise of FedVAEUniCDP (i.e. set $\sigma_G = 5 \cdot 10^{-3}$), we can observe a few percentage points drop in accuracy, but with the benefit of ε halving in the scaled population setting. In other words, if we use noise $\sigma_G = 5 \cdot 10^{-3}$ in FedVAEUniCDP, then we can achieve a maximum accuracy of approximately 84% with $\varepsilon \simeq 1$ in 30000 clients. On the contrary, when it comes to the Centralized VAE model, the privacy guarantees it offers are very weak and cannot be improved by scaling the population size the way we did with federated VAEs, as the model is Centralized.

When it comes to the Epilepsy dataset, we can see that the non-noisy federated VAE model has an f1-score difference of 8-9 percentage points compared to the centralized VAE model. However, as we add noise to FedVAESepCDP up to a certain point, the f1-score seems to increase instead of decrease. In particular, a noise of $\sigma_G = 1 \cdot 10^{-2}$ increases the f1-score by almost 3 percentage points compared to the non-noisy scenario of FedVAESepCDP. This behavior is attributed the fact that a moderate level of noise may act as a regularizer for the VAEs training, essentially avoiding overfitting, improving generalization and contributing to higher f1-scores. However, as we would expect, a high level of noise destabilizes the training process, as seen when $\sigma_G = 2.5 \cdot 10^{-2}$. Nevertheless, if we run FedVAESepCDP Algorithm on Epilepsy using a noise scale of $\sigma_G = 1 \cdot 10^{-2}$, then we can achieve an F1-score of 85%, while achieving CDP with $\varepsilon \simeq 1.7$ in a population of approximately 10000 clients.



(a) Epoch 1400, $\sigma_G = 0$ (b) Epoch 1400, $\sigma_G = 2.5 \cdot 10^{-3}$ (c) Epoch 1400, $\sigma_G = 5 \cdot 10^{-3}$

Figure 4.5.3: Images generated from FedVAEUniCDP in EMNIST

4.5.2 Comparing different data distribution policies

As we discussed in previous sections, one of the fundamental challenges of FL is that the data among different clients are not IID. Hence, to explore the effect this has on the quality of the generated data, we evaluated our Federated VAEs using different data distribution policies (Uniform, KMEANS and Geometric) which were discussed in section (4.4.1). Just as before, we have plotted the accuracy/f1 score of the student network under different noise regimes. However, this time, we have tested different data distribution policies, as shown in figures (4.5.4) and (4.5.5) for the EMNIST and the Epilepsy dataset.

In these figures, the accuracy/f1 score were calculated on an average of 10 runs of the experiment, and the DP guarantees of the different scenarios are the same as the ones in tables (4.5.3) and (4.5.4). Additionally, in order to facilitate the comparison of the different scenarios, we have also sorted our data distribution schemes according to decreasing maximum accuracy/f1 score in tabular form, as shown in tables (4.5.5) and(4.5.6).

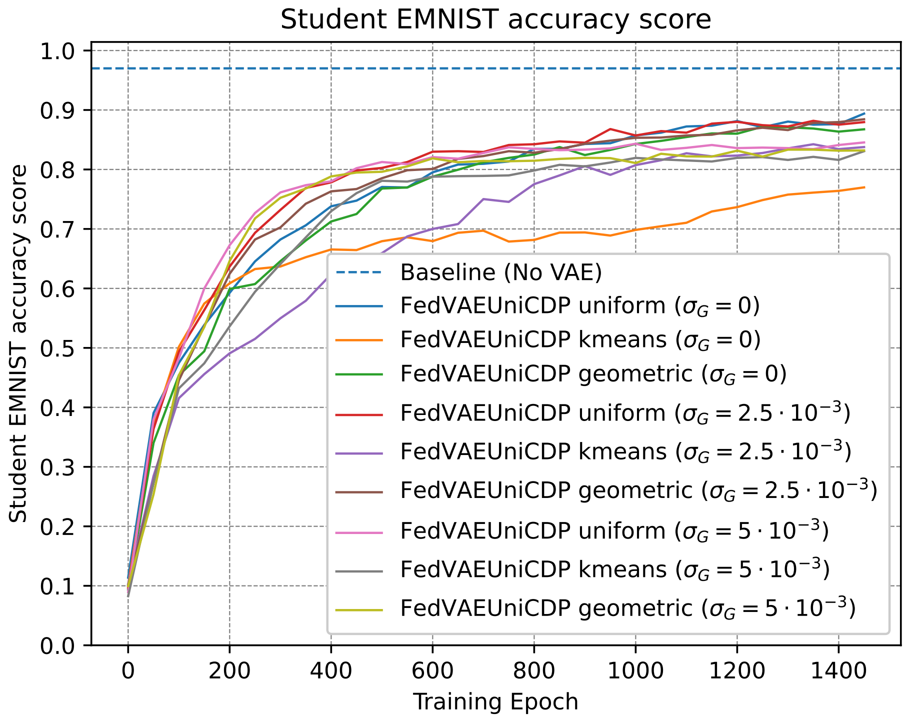


Figure 4.5.4: EMNIST accuracy under different data distribution strategies

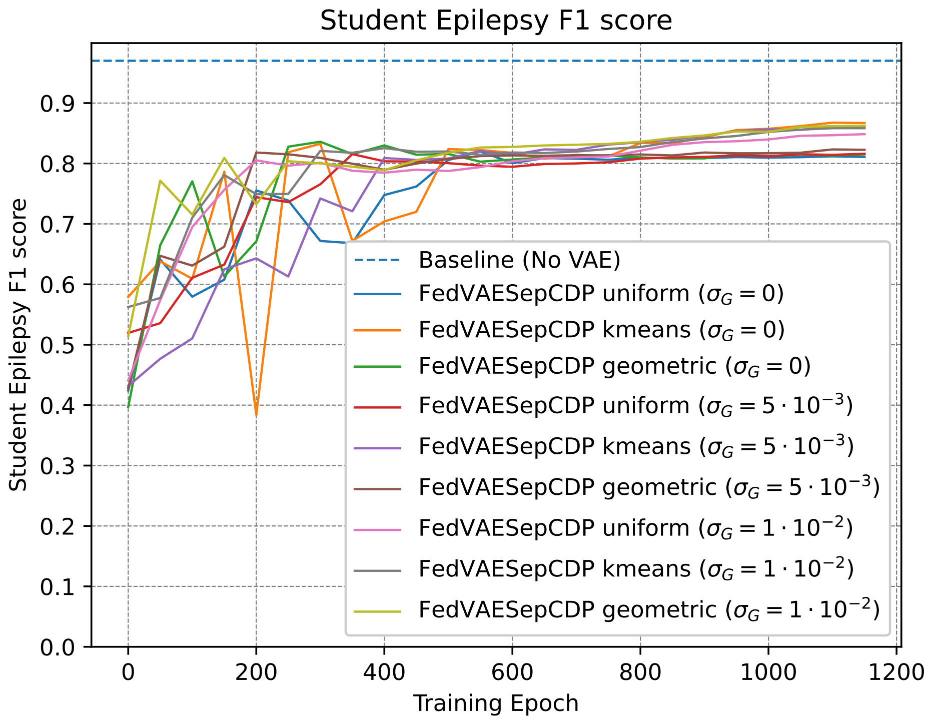


Figure 4.5.5: Epilepsy f1-score under different data distribution strategies

Model	Maximum accuracy
FedVAEUniCDP uniform ($\sigma_G = 0$)	0.894
FedVAEUniCDP geometric ($\sigma_G = 2.5 \cdot 10^{-3}$)	0.884
FedVAEUniCDP uniform ($\sigma_G = 2.5 \cdot 10^{-3}$)	0.882
FedVAEUniCDP geometric ($\sigma_G = 0$)	0.872
FedVAEUniCDP uniform ($\sigma_G = 5 \cdot 10^{-3}$)	0.845
FedVAEUniCDP kmeans ($\sigma_G = 2.5 \cdot 10^{-3}$)	0.842
FedVAEUniCDP geometric ($\sigma_G = 5 \cdot 10^{-3}$)	0.833
FedVAEUniCDP kmeans ($\sigma_G = 5 \cdot 10^{-3}$)	0.831
FedVAEUniCDP kmeans ($\sigma_G = 0$)	0.770

Table 4.5.5: Maximum accuracy in different scenarios of EMNIST

Model	Maximum F1 score
FedVAESepCDP kmeans ($\sigma_G = 0$)	0.868
FedVAESepCDP geometric ($\sigma_G = 1 \cdot 10^{-2}$)	0.862
FedVAESepCDP kmeans ($\sigma_G = 5 \cdot 10^{-3}$)	0.862
FedVAESepCDP kmeans ($\sigma_G = 1 \cdot 10^{-2}$)	0.858
FedVAESepCDP uniform ($\sigma_G = 1 \cdot 10^{-2}$)	0.849
FedVAESepCDP geometric ($\sigma_G = 0$)	0.836
FedVAESepCDP geometric ($\sigma_G = 5 \cdot 10^{-3}$)	0.823
FedVAESepCDP uniform ($\sigma_G = 0$)	0.820
FedVAESepCDP uniform ($\sigma_G = 5 \cdot 10^{-3}$)	0.816

Table 4.5.6: Maximum f1-score in different scenarios of Epilepsy

By looking at Table (4.5.5), we can see that the uniform data distribution strategy without noise seems to generate the best accuracy results for EMNIST. The reason for this is that the uniform distribution strategy ensures i.i.d samples in the federated process.

Hence, since the distribution of samples among clients is not skewed, we expect this to have a positive impact on training. In contrast, the kmeans data distribution seems to perform worse, as it generates clients with very highly imbalanced data. In particular, kmeans results in few clients having many data points and most clients having very few data points. Hence, given that every client is treated equally in our training process (i.e. all updates have the same weight), then the updates of the many low-utility clients (i.e. clients having little data) may negatively influence training. Lastly, the geometric distribution seems to be between the uniform and the KMEANS with respect to accuracy, as it is more skewed than the uniform distribution, but less skewed than the KMEANS distribution.

On the contrary, by looking at Table (4.5.6), we can see that in the Epilepsy dataset, the distribution strategy that yields the best results is kmeans, then the geometric and then the uniform distribution. This behavior may be attributed to the characteristics of the epilepsy dataset itself. For instance, when using KMEANS on Epilepsy, the clients which have a lot of data (as seen in Figure (4.4.2b)), may be able to train their private encoders very well. Hence, since those encoders remain private and are NOT updated by the federated process (only the decoders are updated), they become much better every time they are trained using the large volume of local data that high-utility clients have. This leads to some clients having very rich private encoders, which may positively affect the training process.

In light of these, we can conclude that the impact of data distribution on the quality of the generated data is not clear, as it depends strongly on the underlying dataset used, and probably on whether we use 1 VAE for all classes (as we did with EMNIST) or 1 VAE for every class (as we did with Epilepsy).

4.5.3 Accuracy vs number of parameters

Since VAEs are Neural Networks, it would be reasonable to study the relationship between the number of network parameters (e.g. weights and biases) and the quality of the generated data. For this reason, we experimented with different neural network sizes and different noise regimes, while measuring the accuracy/f1-score of the student network in as shown in figures (4.5.6) and (4.5.7). At this point, it should be noted that when it comes to EMNIST, we measured the total parameters of the 1 VAE we used (in EMNIST we use 1 VAE for all classes), but when it comes to Epilepsy, we measured the sum of parameters of the 2 VAEs we used (in Epilepsy we used 1 VAE for each one of the 2 classes).

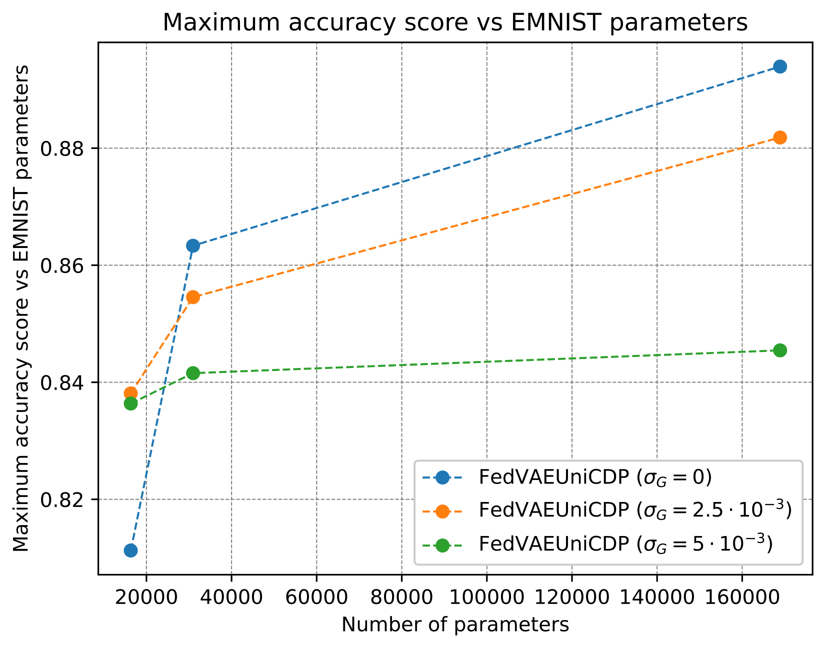


Figure 4.5.6: EMNIST accuracy vs parameter number

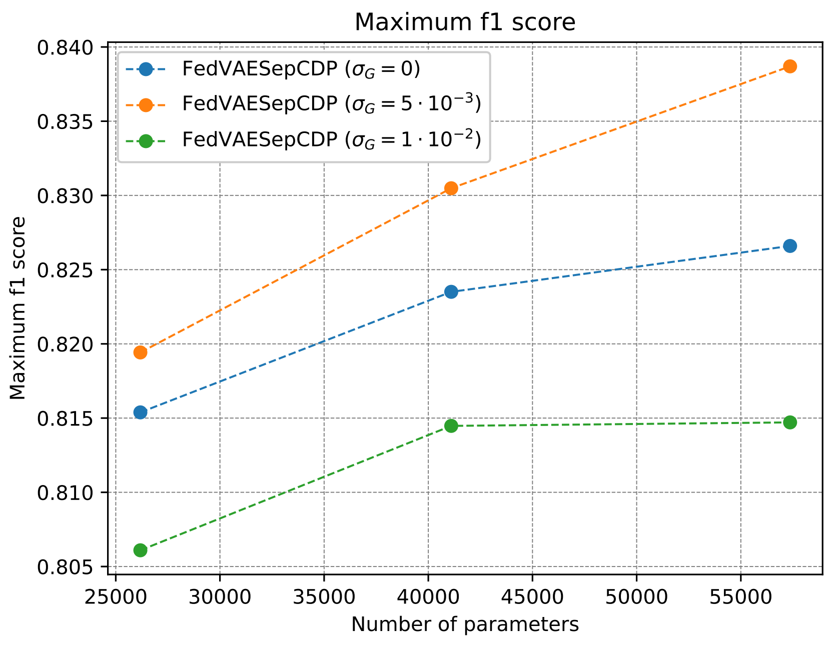


Figure 4.5.7: Epilepsy f1 score vs parameter number

By looking at the 2 plots above, we can see that reducing the number of parameters, the accuracy/f1 score decreases. This behavior is to be expected, as a reduction in the model parameters, generally leads to a poorer modeling of the distributions of our samples, thus resulting in lower-fidelity artificial samples. However, as we can see, if we increase the level of noise, accuracy/f1-score drop at a slower pace when reducing the number of parameters. This may be a result of regularization, as the more noise we add, the less impact the parameters of the neural network have on the final result. For instance, if we add too much noise, the output samples generated by the VAE will be of the same poor quality, no matter how many parameters the network has.

4.5.4 Comparison with PrivBayes

In order to see how well our generative model performs, we compared it with another method called PrivBayes that tries to solve the same problem as the one we are solving. In particular, PrivBayes was adapted to a federated setting by V. Digalakis and C. Zacharioudakis in [54] and [18] and can be used for differentially private data generation from distributed datasets.

For our comparison, we adjusted and optimized the code provided by [18] in order to be able to run federated PrivBayes on the EMNIST and the Epilepsy datasets. The main challenge of running PrivBayes is that it is very computationally expensive, especially for bigger datasets. For instance, in the EMNIST dataset, which consists of samples with 28^2 features, federated PrivBayes takes almost 24 hours to run, even if we pick the smallest possible degree for our Bayesian Network ($k = 1$).

In light of all these, we used a degree of $k = 1$ for all of our experiments and we then ran the PrivBayes algorithm under different noise regimes and data distribution policies. In particular, we tested PrivBayes under different values of ε ($\varepsilon = \infty, \varepsilon = 10^3$ and $\varepsilon = 1$) and under different data distribution schemes (kmeans,geometric and uniform). Then, we evaluated the quality of the generated data using classification accuracy/f1 score as our metrics (accuracy was used for EMNIST and F1-score was used for the Epilepsy dataset). The accuracy/f1 score was evaluated by 10 different classifiers (XGBoost, Decision Tree, KNN, Linear SVC, SVC,Random Forest, Multi-Layer Perceptron, Ada-Boost, Gradient-Boosting, Gaussian NB, Linear Discriminant Analysis and Quadratic Discriminant Analysis).

Then, for each dataset, we picked the best classifier and recorded its accuracy/f1 score in the tables (4.5.7) and (4.5.8) for the EMNIST and the Epilepsy datasets respectively. In the same tables we also recorded the DP guarantees achieved in the different settings, where a value of $\varepsilon = \infty$ corresponds to no differential privacy.

Last but not least, for the actual comparison with our VAEs, we created figures (4.5.8) and (4.5.9). In those figures, we have plotted the accuracy/f1 score of PrivBayes against different privacy budgets and under different data distribution policies. Then, we also plotted the accuracy/f1 scores of our VAEs on a scaled population size of $\sim 10^4$ clients. At this point, we should note that the experiments with PrivBayes were performed under the same federated setting as the VAE experiments (i.e. same dataset, same number of clients $N = 20$, same distribution policy in clients). However, during the comparison in (4.5.8) and (4.5.9), the type of DP guarantees PrivBayes and VAEs offer are different: PrivBayes offers local ε-DP out of the box, while VAEs offer (ε, δ)-CDP which assumes either a trusted server or a secure aggregation protocol. Also,unlike PrivBayes, federated VAEs need around $\sim 10^4$ clients in order to achieve single-digit DP, despite the fact that 10^4 clients are not that many for a real-world FL scenario (in cross-device FL, the number of devices can reach even 10^{10}).

Model	DP (ε, δ)	Accuracy of best classifier
EMNIST centralized (baseline)	-	0.897
PrivBayes UNIFORM	$(\infty, 0)$	0.723
PrivBayes UNIFORM	$(1 \cdot 10^3, 0)$	0.123
PrivBayes UNIFORM	$(1 \cdot 10^0, 0)$	0.128
PrivBayes KMEANS	$(\infty, 0)$	0.830
PrivBayes KMEANS	$(1 \cdot 10^3, 0)$	0.513
PrivBayes KMEANS	$(1 \cdot 10^0, 0)$	0.114
PrivBayes GEOMETRIC	$(\infty, 0)$	0.732
PrivBayes GEOMETRIC	$(1 \cdot 10^3, 0)$	0.175
PrivBayes GEOMETRIC	$(1 \cdot 10^0, 0)$	0.082

Table 4.5.7: Accuracy and privacy for PrivBayes in EMNIST

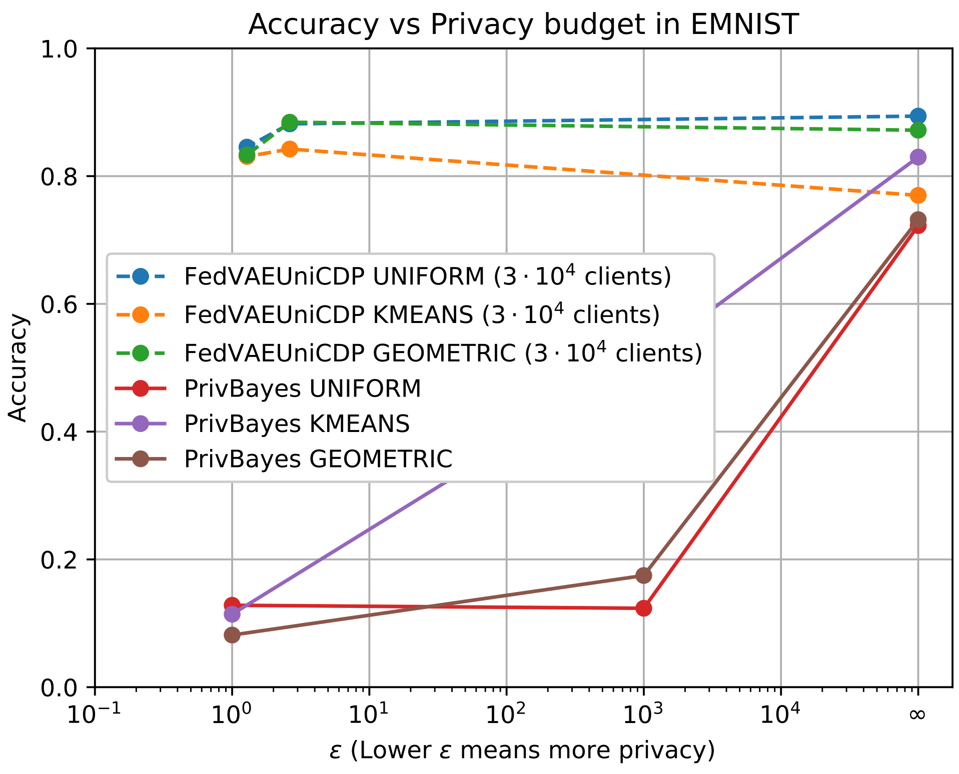


Figure 4.5.8: PrivBayes vs Federated VAEs in EMNIST

Model	DP (ε, δ)	F1 score of best classifier
Epilepsy centralized (baseline)	-	0.940
PrivBayes UNIFORM	$(\infty, 0)$	0.813
PrivBayes UNIFORM	$(1 \cdot 10^3, 0)$	0.836
PrivBayes UNIFORM	$(1 \cdot 10^0, 0)$	0.712
PrivBayes KMEANS	$(\infty, 0)$	0.812
PrivBayes KMEANS	$(1 \cdot 10^3, 0)$	0.779
PrivBayes KMEANS	$(1 \cdot 10^0, 0)$	0.133
PrivBayes GEOMETRIC	$(\infty, 0)$	0.805
PrivBayes GEOMETRIC	$(1 \cdot 10^3, 0)$	0.812
PrivBayes GEOMETRIC	$(1 \cdot 10^0, 0)$	0.621

Table 4.5.8: Accuracy and privacy for PrivBayes in Epilepsy

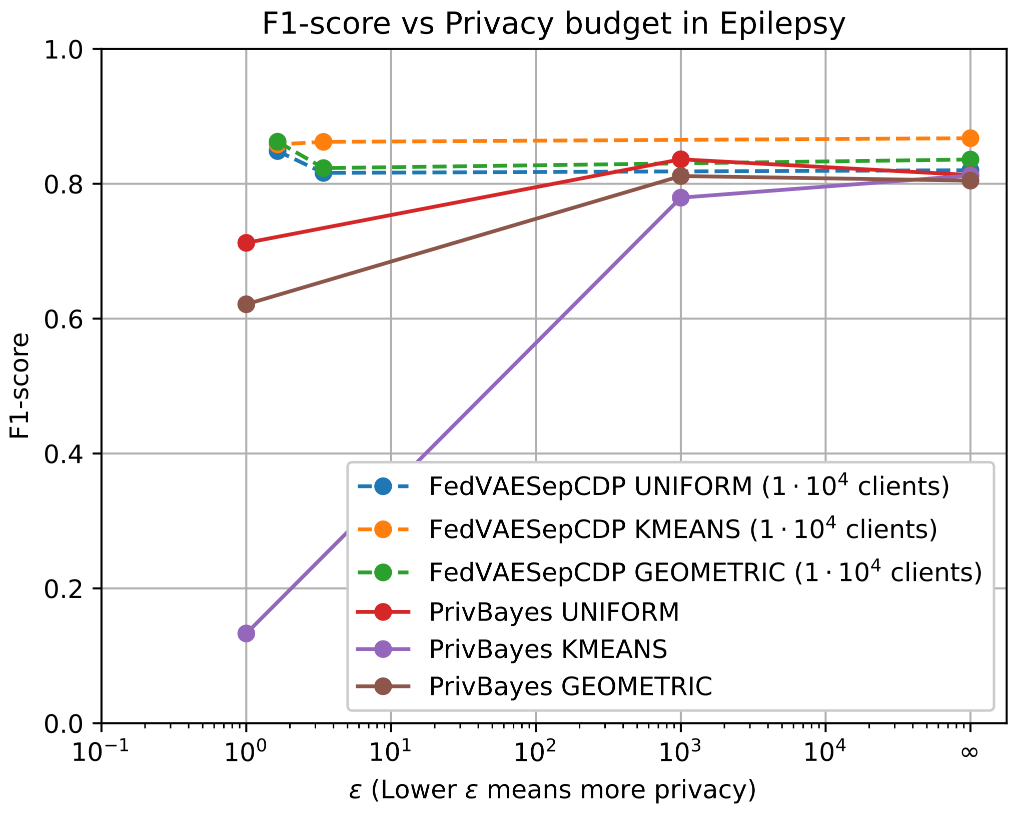


Figure 4.5.9: PrivBayes vs Federated VAEs in Epilepsy

By looking at figure (4.5.8), it is evident that PrivBayes demonstrates very poor performance on the EMNIST dataset, especially in higher levels of noise. In fact, the performance of PrivBayes when $\varepsilon = 1$ is no better than a classifier which selects every class of EMNIST uniformly at random without taking into account the data at all. Hence, the data generated by PrivBayes on EMNIST are equivalent to random noise. The reason for this is that PrivBayes is inherently not suitable for reproducing image datasets, as it doesn't take into account the spatial relationships of the different pixels of the image. In fact, in order to capture those relationships, we would have to use a very high degree k for the Bayesian Network, which would lead to an explosion in complexity. For instance, in the experiments we did, even for $k = 1$, PrivBayes took almost a day to run on EMNIST, and increasing k beyond 1 would lead to an exponential increase in computational time. Hence, in light of all these, when comparing Federated VAEs with PrivBayes on EMNIST, VAEs seem to demonstrate a better performance, under the assumption that we have $3 \cdot 10^4$ clients at our disposal. Hence, if this condition is met, then VAEs can achieve single-digit ε on EMNIST with $\sim 83\%$ accuracy, while PrivBayes achieves roughly the same guarantees with 10% accuracy on average.

Then, by examining figure 4.5.9, it is evident that for $\varepsilon = 1$, PrivBayes demonstrates mediocre performance for Uniform and Geometric distributions and very poor performance for KMEANS distribution. However, the performance of PrivBayes in Epilepsy is clearly better than the performance in EMNIST, as PrivBayes is much better suited for synthesizing tabular datasets than image datasets. In any case, Federated VAEs seem to win PrivBayes on the Epilepsy dataset as well, under the assumption that we have a client population of around 10^4 clients or more. In particular, PrivBayes achieves single-digit ε with a maximum f1-score of $\sim 70\%$ on uniform data distributions of Epilepsy, while Federated VAEs achieve an f1-score of around 85%, while maintaining single-digit ε in 10^4 clients.

At this point, we should note again that although Federated VAEs seem to achieve better performance than PrivBayes on both EMNIST and Epilepsy, Federated VAEs make more assumption than PrivBayes. In particular, the DP guarantees given by Federated VAEs assume the existence of a trusted server or a secure aggregation protocol, and at the same time, the participation of around 10^4 clients. On the other hand, PrivBayes doesn't make such assumptions and offers $(\varepsilon, 0)$ differential privacy for a small number of clients. At the same time, the limited scope of our experiments doesn't allow us to generalize our conclusions regarding the comparative performance of Federated VAEs and PrivBayes. In particular, in order to perform a more rigorous comparison of Federated VAEs with PrivBayes, we would need to experiment with much more datasets (tabular and image ones), with different degrees for the Bayesian network, different strategies of constructing the Bayesian Network described in [54] (i.e. sharing the noisy sufficient statistics, noisy majority voting, sharing the noise model) and probably with different classifiers to estimate the quality of the synthetic data generated by PrivBayes.

4.5.5 Comparison with GANs

Since GANs are considered to be state-of-the-art models when it comes to data generation, it would be very reasonable to compare our VAEs against GANs in a federated and DP setting. In order to run a fair comparison against GANs, we will rely on the work done by Augenstein et al. [13], which we described in Section (4.2.1). Using this work and its implementation [30] as our reference, we adapted GANs to our federated DP setting and ran experiments with the EMNIST dataset under different noise regimes. At this point, it should be noted that in order to use DPFedGans in our setting, we trained 10 GANs, 1 for every class of EMNIST, where the i-th GAN was trained using the subset of the data that corresponds to the i-th class. Then, we used Algorithm (3) for the actual training process. The parameters we used for Federated DP GANs are similar to the ones we used for Federated DP VAEs and can be shown in table (4.5.9).

Using the setting we described, we trained Federated DP GANs on EMNIST, while using the Student network to assess the quality of the generated data. Just like with Federated VAEs, the student network was trained on artificial data and evaluated on the real test set. Then, we plotted the accuracy of the student network every 50 federated rounds, as shown on Figure (4.5.10). Also, we ran Algorithm (3) under different noise levels ($\sigma_G = 0$, $\sigma_G = 2.5 \cdot 10^{-3}$ and $\sigma_G = 5 \cdot 10^{-3}$) and then calculated the DP guarantees of DPFedGANs in simulated and scaled populations, as shown in table (4.5.10). Then, for the actual comparison of Federated GANs with Federated DP VAEs, we created Radar plot (4.5.11) in order to facilitate comparison across 4 different factors. In particular, we compared our FedVAEUniCDP against DPFedGAN across the following dimensions:

- **Privacy budget in $3 \cdot 10^4$ clients**: In section (4.5.1), we found out that if we use $N = 30000$ clients, then the FedVAEUniCDP algorithm can achieve single-digit ε CDP guarantees on EMNIST, without significantly compromising utility. Hence, when comparing Federated DP VAEs with GANs, we will compare their respective privacy budgets ε in a scaled-up population scenario where $\delta = 1/N^{1.1}$, $N = 30000$ and the notion of Differential Privacy used is CDP.

- **Student Accuracy**: This metric refers to the accuracy of the student network when trained on artificial data and then tested on the real test set. Student accuracy can be used to assess the quality of the synthetic datasets generated from the federated VAEs and GANs. However, for more concrete results, student accuracy was calculated by averaging multiple runs of Federated DP GANs and VAEs.

- **Number of Network Parameters**: Since GANs and VAEs are both neural networks, one other metric we can use when comparing them is the number of network parameters that each one of these model requires. Neural Network parameters can be either trainable parameters, such as weights and biases trained through backpropagation, or non-trainable parameters, such as the parameters of Batch Normalization layers which are not trained through backpropagation. Hence, when it comes to FedVAEUniCDP, we counted the number of total parameters of the 1 encoder and the 1 decoder we used (we remind the reader that the FedVAEUniCDP algorithm has 1 VAE for all classes). On the other hand, when it comes to Federated GANs, despite the fact that we used 10 GANs in total (i.e. 1 for every class), we accounted for the parameters of just 1 GAN (i.e 1 Generator and 1 Discriminator), in order to make the comparison more fair.

- **Training time**: The last metric we will use when comparing VAEs to GANs is the time it takes for our federated model to train. When it comes to training time, it should be noted that all the experiments were run in a computer with an NVIDIA GPU, which significantly accelerates training.

gen. learning rate η_G	disc. rate η_D	latent dim n_U	batch size B
$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	30	10
L2 clip parameter S	total users N	users per round qN	federated rounds T
1	10	5	1400
noise scale z	data distribution	class seperation	
$0,\ 0.01,\ 0.02$	UNIFORM	1 GAN for every class	

Table 4.5.9: Hyperparameters for EMNIST Federated DP GANs

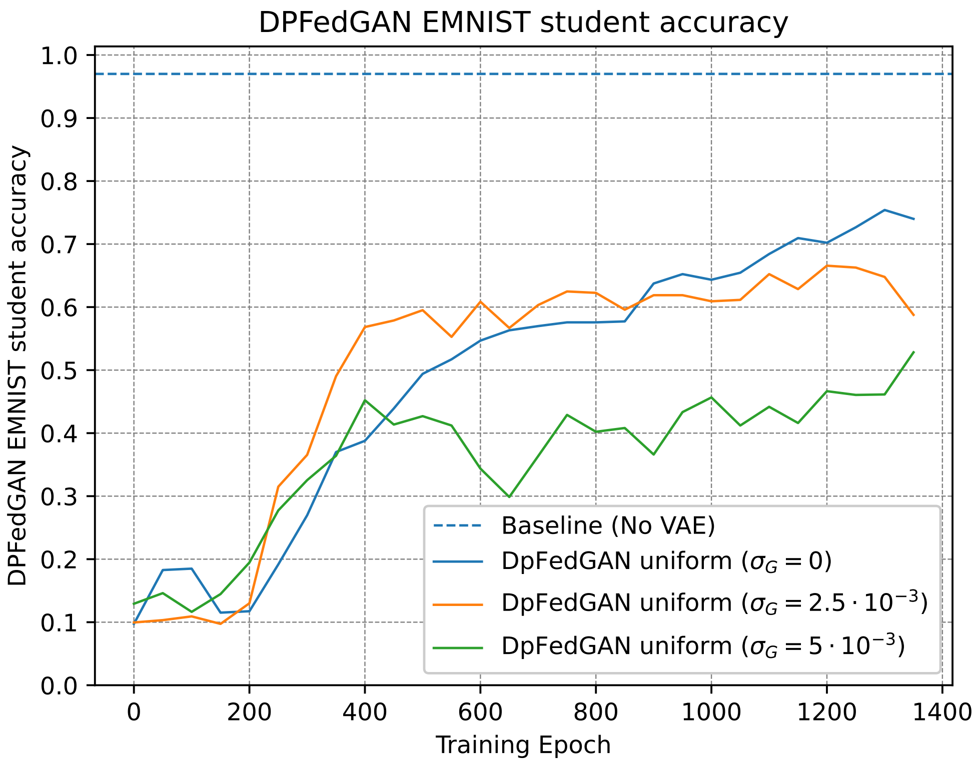


Figure 4.5.10: DPFedGAN Student Accuracy on EMNIST

Model	DP in simulation (N=8)	DP in scaled population (N=30000)
DpFedGAN ($\sigma_G = 0$)	$(\infty, 1.02 \cdot 10^{-1})$	$(\infty, 1.19 \cdot 10^{-5})$
DpFedGAN ($\sigma_G = 2.5 \cdot 10^{-3}$)	$(1.40 \cdot 10^{7}, 1.02 \cdot 10^{-1})$	$(5.33 \cdot 10^{0}, 1.19 \cdot 10^{-5})$
DpFedGAN ($\sigma_G = 5 \cdot 10^{-3}$)	$(3.50 \cdot 10^{6}, 1.02 \cdot 10^{-1})$	$(2.54 \cdot 10^{0}, 1.19 \cdot 10^{-5})$

Table 4.5.10: Privacy budget of DPFedGANs on EMNIST

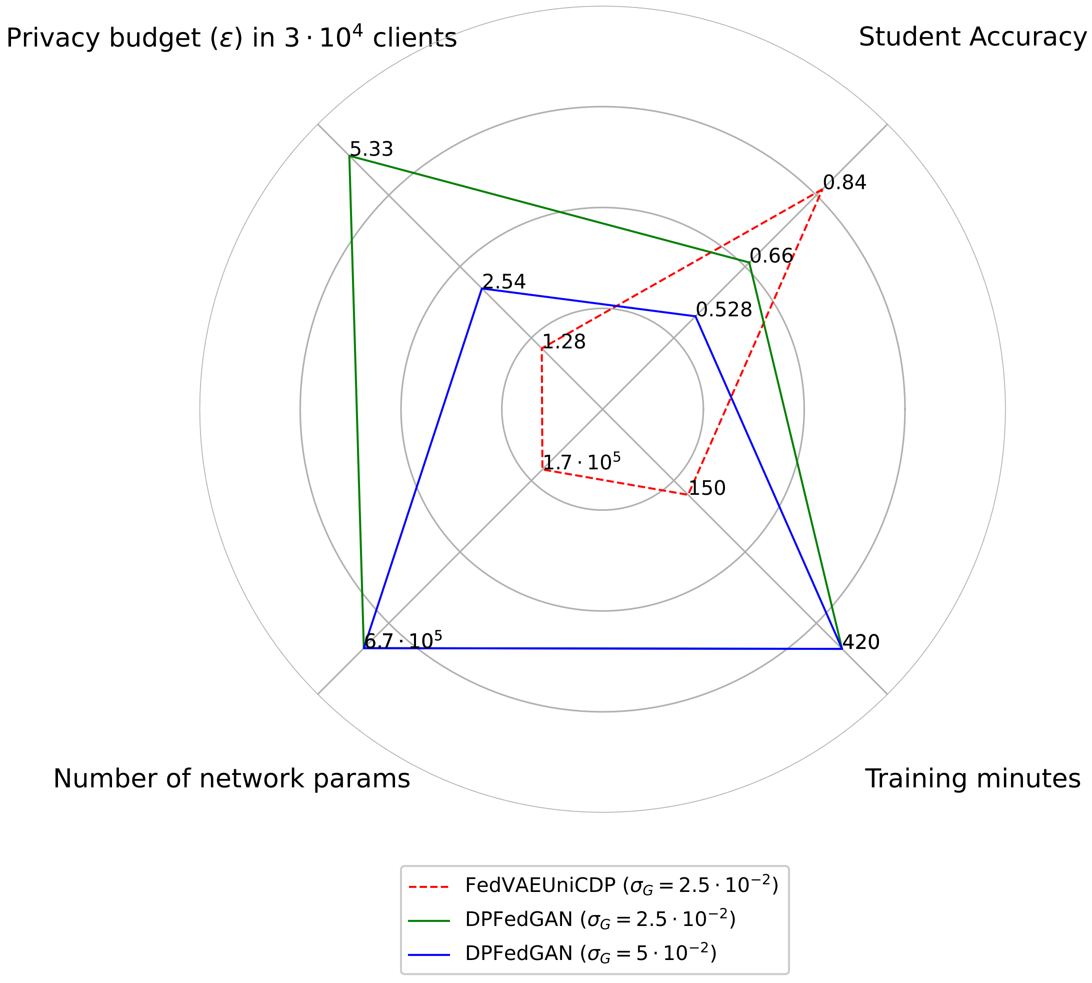


Figure 4.5.11: Comparison of Federated GANs and VAEs

By looking at figure (4.5.10), we can see that a non-noisy DPFedGan achieves a maximum accuracy of around 75%. In contrast, the noisy FedGAN with $\sigma_G = 2.5 \cdot 10^{-3}$ achieves a maximum accuracy of $\sim$ 67% with a CDP privacy budget of 5.3 in 30000 clients. Lastly, the noisy FedGAN with $\sigma_G = 5 \cdot 10^{-3}$ achieves a maximum accuracy of $\sim$ 53% and a privacy budget of 2.5 in 30000 clients. However, it should be noted that in this noisy scenario, the accuracy plot is not very stable.

Additionally, by examining figure (4.5.11), it seems that within the limited scope of our experiments, Federated VAEs appear to win Federated GANs in every comparison dimension of EMNIST. In particular, our Federated VAE achieved 84% accuracy with $\varepsilon = 1.28$ in $3 \cdot 10^4$ clients, while the most noisy DPFedGAN achieved twice the privacy budget and $\sim$ 52% accuracy with 4 times as many parameters as the Federated VAEs. Hence, in our experiments, Federated VAEs seem to demonstrate a better performance than Federated GANs.

However, it is very important to note that since the scope of our experiments is very limited, we cannot draw general conclusions about the performance of VAEs versus the performance of GANs. In particular, generalizing our results would require experimentation with many different datasets besides EMNIST, with different client population sizes, different neural network architectures for VAEs and GANs (e.g. different size of the latent space), and potentially different architectures for the student network. At the same time, we should note that in our experiments, despite the fact that we used the same student network architecture for VAEs and GANs, additional fine-tuning of the Federated GANs may have improved the GAN results further, thus reducing the accuracy and privacy gaps between VAEs and GANs.

4.5.6 Local vs global DP

In all of our experiments so far, we have only experimented with Central Differential Privacy, using Algorithms (4) and (6) for our Federated VAEs. Hence, our assumption has been that we have access to a secure aggregation protocol (e.g. MPC) which allows us to ensure CDP under the presence of an honest-but-curious server. However, since access to such protocols is not always possible, we will now experiment with LDP, while comparing it with the CDP settings we tested. In particular, in order to compare LDP with the CDP settings of our previous experiments, we will attempt to measure the number of clients that the LDP settings requires to (a) maintain the same utility as the CDP setting and (b) ensure single-digit ε guarantees for the average client [7]. Then, we will compare the number of clients needed to achieve single-digit LDP with the number of clients needed to achieve single digit CDP.

In particular, according to Theorem (4.1), in order to compare the privacy guarantees of CDP and LDP against different population sizes and under the same utility (i.e. noise level), we can work as follows:

1. Run Algorithm FedVAESepCDP or FedVAEUniCDP to determine the maximum std of global noise σ_G that can be added while maintaining acceptable utility.
2. For different user population sizes N:
   (a) Initialize a Moments Accountant M_G for the CDP case and a Moments Accountant M_L for the LDP case.
   (b) Call T times the method $\mathcal{M}_G$.compose_subsampled_machanism($\bar{\sigma}, q$) with $\bar{\sigma} = 1$ and $q = S/(\sigma_G \cdot N)$. Then call $\mathcal{M}_G$.get_epsilon(δ) with $\delta = 1/N^{1.1}$ to get the CDP privacy budget of Algorithm FedVAESepCDP or FedVAEUniCDP. This privacy budget represents the CDP privacy budget in a scaled population of size N where the noise level σ_G remains constant (i.e. utility is maintained).
   (c) Call qT times the method $\mathcal{M}_L$.compose_subsampled_machanism($\bar{\sigma}, q$) with $\bar{\sigma} = (\sigma_G\sqrt{qN})/S$. Then call $\mathcal{M}_L$.get_epsilon(δ) with $\delta = 1/N^{1.1}$ to get the average LDP privacy budget of FedVAESepLDP or FedVAEUniLDP. This privacy budget represents the privacy budget of the average client in a scaled LDP setting of N clients. Additionally, this LDP setting has the same utility as the CDP setting above, as the local Gaussian noise with $\sigma_L = \sigma_G\sqrt{qN}$ added by the clients is exactly equivalent to a global Gaussian noise with std σ_G (see Theorem (4.1)).
3. Plot the privacy budgets ε for the CDP and the LDP that were calculated in the previous step for different user populations N.

In order to better illustrate this process, let's consider an example. Let's assume that after running FedVAEUniCDP on EMNIST, we have concluded that a central noise with

[7]When we refer to an *average* client, we mean that this client participates in at most qT training rounds, which is the average number of rounds that a client will participate, since the probability that a client participates in a given round is q.

std $\sigma_G = 5 \cdot 10^{-3}$ maintains an acceptable level of accuracy. Then, using the comparison process we described above, we will attempt to answer the following questions:

- If we maintain the same utility (i.e. keep σ_G constant), then how many clients do we need in order to guarantee single-digit ε CDP?
- If we perform the same experiment in an LDP setting, then how many client do we need so that (a) the total noise added by the clients locally is equivalent to a global noise with std σ_G, and (b) the average client achieves (ε, δ)-LDP with single digit ε.

Hence, in order to explore the relationship of CDP, LDP, privacy budget and population size, we created Figures (4.5.12) and (4.5.13) for the EMNIST and the Epilepsy dataset respectively, by following the process we described previously. In particular, in Figure (4.5.12) we can see the privacy budget in a CDP and an LDP setting for EMNIST under different population sizes, wheres in Figure (4.5.12) we can see the same results, but for the Epilepsy dataset. Also, we should note that the noise level we used in the CDP scenarios of those plots is the same as the noise level we used during our CDP experiments (i.e. Figure (4.5.1) and Figure (4.5.2)). Similarly, the local noise we used in the LDP scenarios of the plots is equivalent to a global noise with std σ_G. Hence, whenever we use σ_G within the context of an LDP scenario, we are implying that this LDP scenario has a local noise which is equivalent to a global noise with std σ_G. If we take that into account, we can see that the green line of every plot has the same utility as the red line, and the blue line has the same utility as the orange line.

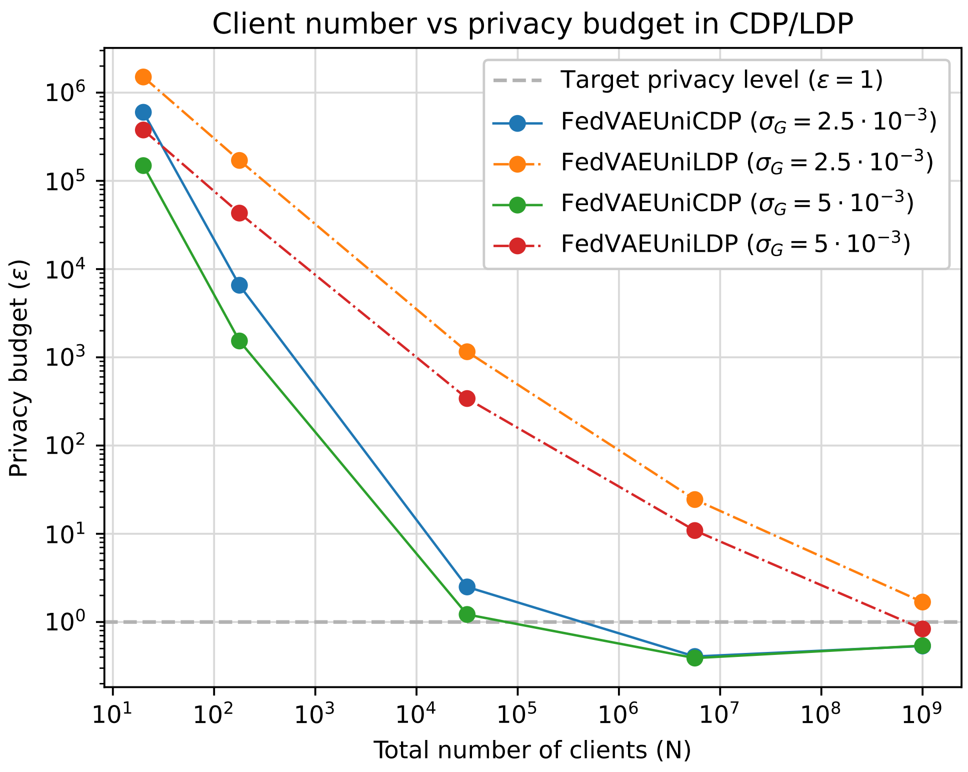


Figure 4.5.12: Client number vs Privacy trade-off in EMNIST

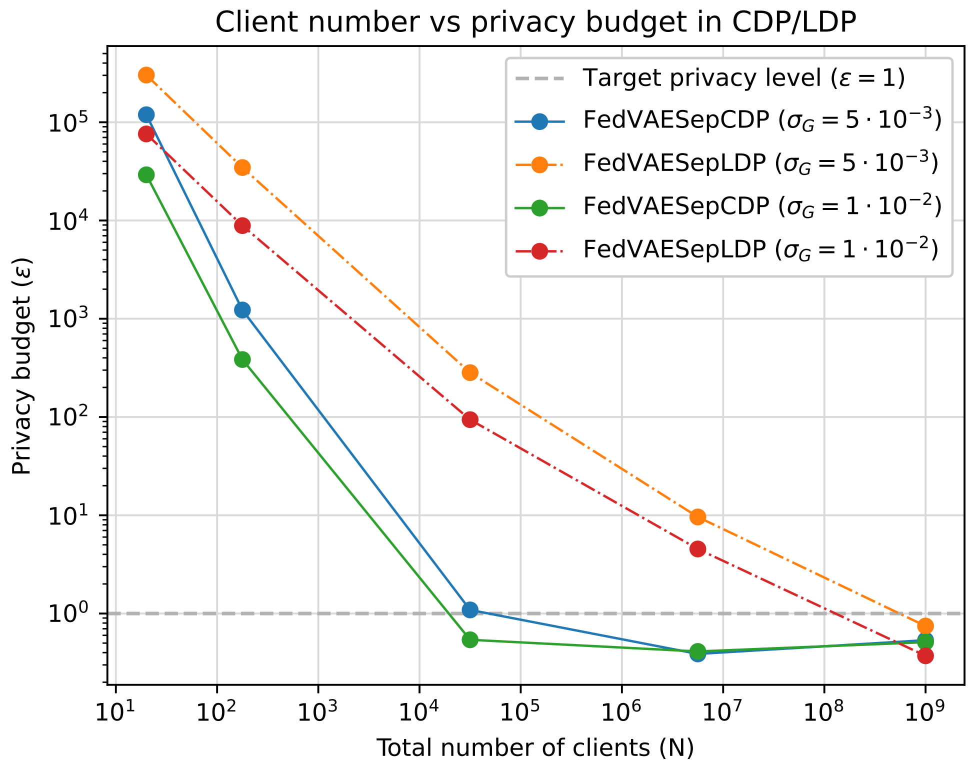


Figure 4.5.13: Client number vs Privacy trade-off in Epilepsy

By examining figure (4.5.12), we can see that we need between 10^4 and 10^5 clients to achieve (ε, δ)-CDP with single-digit ε on EMNIST, when $\sigma_G = 5 \cdot 10^{-3}$. That's why in Table (4.5.3) we picked $N = 30000$ for the scaled population size. However, when it comes to LDP, the number of clients needed to achieve equivalent level of privacy is much greater. In particular, in order to achieve the same utility as CDP (i.e. $\sigma_G = 5 \cdot 10^{-3}$), while ensuring single-digit LDP to the average user, we would need around 10^8 clients on EMNIST.

Similarly, by looking at Figure (4.5.13) for Epilepsy, we can see that for a noise level of $\sigma_G = 1 \cdot 10^{-2}$, we need $\sim 10^4$ clients for CDP with single-digit ε and $\sim 10^8$ clients for LDP with single-digit ε for the average client.

Chapter 5

Conclusions

To summarize, in this thesis we discussed the problem of differentially private synthetic data generation from distributed datasets.

In the first chapter, we highlighted the necessity for privacy in our data-driven world and briefly defined the nature and importance of the problem we are trying to solve.

In the next chapter, we analyzed the challenges of privacy-preserving analytics and pinpointed why most dataset anonymization techniques have failed. This led us to explore the state-of-the art concept of Differential Privacy, along with its many attractive properties and principles.

In chapter 3 we explored the widely-adopted paradigm of Federated Learning and highlighted its strengths in performing Machine Learning on Decentralized datasets. We then analyzed some of its privacy aspects and highlighted its connection with Differential Privacy and Cryptography.

In the final chapter, we examined the problem of private, synthetic data generation from decentralized datasets. After presenting some related work in the area, we went on to discuss the attractive properties of Generative Models and particularly Variational Autoencoders in solving this problem. Then, we analyzed the principles and mathematical framework of VAEs, while proposing ways to adapt VAEs to a federated and Differentially Private setting. After that,we tested our approach in various experimental settings, by exploring the impact of noise, data distribution and number of neural network parameters on the quality of the synthetic data generated. We also compared our federated model against other methods of solving the same problem, such as PrivBayes and Federated DP GANs. Last but not least, we compared Central Differential Privacy and Local Differential Privacy as different privacy paradigms, while exploring the underlying assumptions that each one of those requires.

5.1 Result Summary & Contribution

As we previously discussed, we addressed the problem of Private Synthetic Data Generation from Decentralized datasets by combining Variational Autoencoders, Federated Learning and Differential Privacy. Our results showed that we can achieve good privacy and utility, while using an FL scenario of $\sim 10^4$ clients in conjunction with a secure aggregation protocol. We also showed that if we don't have access to such protocols, we need $\sim 10^8$ clients to maintain acceptable utility while ensuring good LDP guarantees for the average client.

To our knowledge, our approach is the first that uses VAEs in a federated and DP setting, while ensuring user-level differential privacy. In fact, the federation of DP VAEs with a private decoder for every client, was deemed an open problem by Augenstein et al. [13]. In particular, the reason we chose to address this open problem is that by keeping the decoder private, we can reduce the amount of computation and communication, improve privacy, and potentially improve the quality of synthetic data generated by the VAE.

Also, within the limited scope of our experiments, our approach seems to compare very favorably to approaches like PrivBayes and DPFedGANs, not only with respect to privacy and utility, but also with respect to training time and computational complexity. In particular, approaches like PrivBayes are far from perfect, as they work only on specific types of datasets (i.e. performance in image datasets is very poor), while their computational complexity skyrockets if we increase the number of features or the degree of the Bayesian Network. At the same time, Federated VAEs seem to be more tolerant to noise than Federated GANs. In particular, our experiments showed that under similar settings, Federated VAEs maintain better privacy guarantees than Federated GANs, despite the fact that many more experiments are needed to generalize this conclusion.

At the same time, federated DP VAEs can theoretically be used to synthesize any type of dataset (e.g. image, tabular), depending on the architecture we use for the encoder and the decoder. In that sense, we offer a potentially general approach towards differentially private synthetic data generation from distributed datasets. This approach can be utilized by a group of organizations to jointly create a synthetic dataset using their collective data, without revealing their data to one another. Then, the organizations can use the synthetic dataset to enrich their private datasets and train their models, while being assured that the use of the synthetic dataset by anybody won't leak information about the original data or infringe the privacy of the data owners. Last but not least, the Differentially Private synthetic datasets can also be used for hyperparameter tuning within the context of Federated Learning: If the real data is used for hyperparameter tuning of a model, then each time the experiment is rerun on the real data, the privacy loss will increase. On the contrary, if the differentially private synthetic data are used, then we can try many different hyperparameters and run as many experiments as we want on the synthetic data, without those experiments incurring any additional privacy losses.

5.2 Drawbacks

Despite the promising nature of Federated VAEs, our approach has some disadvantages that make it unsuitable for certain FL scenarios. In particular, our CDP approach relies on the assumption that we have access to a secure aggregation protocol, which may not always be the case. Additionally, even if we have access to such protocols, the minimum number of clients needed to achieve good privacy guarantees is 10^4. This means that our approach cannot be used in Federated scenarios with a limited number of clients, such as cross-silo FL scenarios with few participating organizations.

On the other hand, our LDP approach, doesn't assume the existence of a secure aggregation protocol. However, it requires the participation of around $10^8 - 10^9$ clients, thus being unsuitable for most real-world FL scenarios, with the possible exception of cross-device FL scenarios such as the ones orchestrated by Google on mobile devices (e.g. Google Keyboard). In addition to that, the LDP guarantees we have given are guarantees for the "average" user which participates in at most qT training rounds. Hence, although the average participation rounds of a user are indeed qT, there may be users that participate in more than qT rounds and users that participate in less than qT rounds. This may lead to some clients spending greater privacy budgets than the average, and possibly exceeding their original budgets. This problem, however, can be alleviated if the clients refuse participation in further rounds when they have exceeded their budget (i.e. the server may only select clients which haven't exceeded their privacy budget).

Another drawback of Federated VAEs and VAEs in general is that they tend to become unstable if they are not well parametrized, especially under the presence of noise. For instance, in many cases, VAEs suffer from exploding loss functions, thus requiring careful parametrization and additional clipping in order to achieve a stable training. At the same time, we should also note that although the nature of our results is promising, our experiments had a very limited scope. In fact, in order to generalize our results, we should run much more experiments with different datasets, different data distribution policies, different VAE hyperparameters, different client populations, and examine many other aspects of Federated VAEs which we did not study in this thesis.

5.3 Future Work

Besides the areas we covered, there are still many aspects of Federated VAEs that are worth exploring:

- **Private vs non-private encoder**: In our work, we used a private encoder for our VAEs which wasn't shared with the server (only the decoders were federated). Hence, it would be worth exploring whether federating just the decoder yields better accuracy than federating both the encoder and the decoder.

- **Alternative notions of LDP**: In our approach, we achieved LDP through the addition of carefully calibrated Gaussian noise in the clients. However, this clearly isn't the only way of achieving LDP. In fact, there are also other LDP mechanisms which may offer better privacy and utility guarantees than the Gaussian mechanism we used. For instance, in [56], the authors have developed custom randomized mechanism which is based on Bernoulli random variables and presumably achieves better utility than the Optimal Gaussian mechanism for the same LDP guarantees. Other approaches towards LDP have also been proposed (e.g. [60],[44],[53]), and thus it would be worth exploring if some of those approaches can be combined with Federated VAEs so as to achieve better privacy and utility.

- **Other datasets**: Besides the EMNIST and the Epilepsy datasets we used, it is also useful to evaluate Federated VAEs in other datasets as well. For instance, it would be very beneficial to test VAEs on tabular and image biomedical datasets in order to explore the suitability of VAEs for healthcare applications.

- **Realistic populations and data distributions**: In our experiments, we tested Federated VAEs in a small population of $N = 20$ clients, with a predefined set of data federation strategies (i.e. uniform, kmeans, geometric). However, in order to validate and generalize our results, it is very important to test Federated VAEs in realistic population sizes with real federated data. The reason for this is that the number of clients and the non-IID distribution of data may be important factors in training Federated VAEs, and thus the relationship of those factors with the quality of the trained model should be explored.

- **Evaluate synthetic data differently**: In our experiments, we used the Student Network in order to evaluate the quality of the generated data. This method, however, is not the only metric that can be used: There are many other methods worth exploring. For instance, the MIT synthetic data vault project (Github implementation [55], Publication [46]) uses a combination of many different metrics to evaluate the quality of the synthetic dataset, such as LogisticRegression Detection, SVC Detection, Gaussian Mixture Log Likelihood, Chi-Squared, Inverted Kolmogorov-Smirnov D statistic,Continuous KL Divergence etc.

Bibliography

[1] *K–anonymity*. "https://en.wikipedia.org/wiki/K-anonymity".

[2] *K–anonymity:An Introduction*. "https://www.privitar.com/blog/k-anonymity-an-introduction/".

[3] *Linkage Attacks*. "https://www.privitar.com/glossary/linkage-attack/".

[4] From Autoencoder to Beta-VAE, Aug 2018. [Online; accessed 17. May 2021].

[5] Variational Autoencoder Explained, Oct 2018. [Online; accessed 16. May 2021].

[6] Introducing Variational Autoencoders (in Prose and Code), Apr 2021. [Online; accessed 16. May 2021].

[7] UCI Machine Learning Repository: Epileptic Seizure Recognition Data Set, May 2021. [Online; accessed 20. May 2021].

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

[9] Gergely Acs, Luca Melis, Claude Castelluccia, and Emiliano De Cristofaro. Differentially Private Mixture of Generative Neural Networks. *arXiv*, Sep 2017.

[10] Jinwon An and S. Cho. Variational Autoencoder based Anomaly Detection using Reconstruction Probability, 2015. [Online; accessed 12. May 2021].

[11] Ralph Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Phys. Rev. E*, 64(6 Pt 1):061907, Jan 2002.

[12] A. Asperti, D. Evangelista, and E. Loli Piccolomini. A survey on Variational Autoencoders from a GreenAI perspective. *arXiv*, Mar 2021.

[13] Sean Augenstein, H. Brendan McMahan, Daniel Ramage, Swaroop Ramaswamy, Peter Kairouz, Mingqing Chen, Rajiv Mathews, and Blaise Aguera y Arcas. Generative models for effective ml on private, decentralized datasets, 2020.

[14] TensorFlow Authors. *Convolutional Variational Autoencoders*, 2020. "https://www.tensorflow.org/tutorials/generative/cvae".

[15] Pierre Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49. JMLR Workshop and Conference Proceedings, Jun 2012.

[16] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. Association for Computing Machinery, New York, NY, USA, Oct 2017.

[17] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure Computation for Machine Learning With SPDZ. *arXiv*, Jan 2019.

[18] Zacharioudakis Christos. Large differentially private data synthesis, 2020. Diploma Work, School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, 2020 "https://doi.org/10.26233/heallink.tuc.84556".

[19] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[20] T. Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15(429-444):2–1, 1977.

[21] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[22] Andrew DeCotiis-Mauro. *Intuitively Understanding Variational Autoencoders*, 2018. "https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf".

[23] C. Dwork and R. Pottenger. Toward practicing privacy. *J Am Med Inform Assoc*, 20(1):102–108, Jan 2013.

[24] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60, 2010.

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

[26] Thanos Giannopoulos and Dimitris Mouris. Privacy Preserving Medical Data Analytics using Secure Multi Party Computation. An End-To-End Use Case. *ResearchGate*, Sep 2018.

[27] Olga Gkountouna. A survey on privacy preservation methods. 2015.

[28] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, April 1988.

[29] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv*, Jun 2014.

[30] google research. federated, May 2021. [Online; accessed 29. May 2021].

[31] Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. MD-GAN: Multi-Discriminator Generative Adversarial Networks for Distributed Datasets. *arXiv*, Nov 2018.

[32] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, Jul 2006.

[33] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2021.

[34] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv*, Dec 2013.

[35] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE J.*, 37(2):233–243, Feb 1991.

[36] Xu Lei, Skoularidou Maria, Cuesta-Infante Alfredo, and Veeramachaneni Kalyan. Modeling Tabular data using Conditional GAN. *arXiv*, Jul 2019.

[37] N. Li, T. Li, and S. Venkatasubramanian. Closeness: A new privacy measure for data publishing. *IEEE Transactions on Knowledge and Data Engineering*, 22(7):943–956, 2010.

[38] Xiaopeng Li and James She. Collaborative Variational Autoencoder for Recommender Systems. In *KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 305–314. Association for Computing Machinery, New York, NY, USA, Aug 2017.

[39] Jaechang Lim, Seongok Ryu, Jin Woo Kim, and Woo Youn Kim. Molecular generative model based on conditional variational autoencoder for de novo molecular design. *J. Cheminf.*, 10(1):1–9, Dec 2018.

[40] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. L-diversity: privacy beyond k-anonymity. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 24–24, 2006.

[41] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), March 2007.

[42] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.

[43] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models, 2018.

[44] Thông T. Nguyên, Xiaokui Xiao, Yin Yang, Siu Cheung Hui, Hyejin Shin, and Junbum Shin. Collecting and Analyzing Data from Smart Device Users with Local Differential Privacy. *arXiv*, Jun 2016.

[45] Achraf Oussidi and Azeddine Elhassouny. Deep generative models: Survey. In *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–8. IEEE, Apr 2018.

[46] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The Synthetic Data Vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410. IEEE, Oct 2016.

[47] Andrew J. Paverd and A. Martin. Modelling and Automatically Analysing Privacy Properties for Honest-but-Curious Adversaries, 2014. [Online; accessed 19. May 2021].

[48] Rafael Pinot. Minimum spanning tree release under differential privacy constraints. 01 2018.

[49] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. 1 1998.

[50] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[51] Tsubasa Takahashi, Shun Takagi, Hajime Ono, and Tatsuya Komatsu. Differentially Private Variational Autoencoders with Term-wise Gradient Aggregation. *arXiv*, Jun 2020.

[52] Reihaneh Torkzadehmahani, Peter Kairouz, and Benedict Paten. DP-CGAN: Differentially Private Synthetic Data and Label Generation. *arXiv*, Jan 2020.

[53] Stacey Truex, Ling Liu, Ka-Ho Chow, Mehmet Emre Gursoy, and Wenqi Wei. LDP-Fed: Federated Learning with Local Differential Privacy. *arXiv*, Jun 2020.

[54] Digalakis Vasileios. Data analytics with differential privacy, 2018. Diploma Work, School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, 2018 "`https://doi.org/10.26233/heallink.tuc.78371`".

[55] MIT Synthetic Data Vault. SDV, Jun 2021. [Online; accessed 24. Jun. 2021,"`https://github.com/sdv-dev/SDV`"].

[56] Teng Wang, Jun Zhao, Xinyu Yang, and Xuebin Ren. Locally Differentially Private Data Collection and Analysis. *arXiv*, Jun 2019.

[57] Yu-Xiang Wang, Borja Balle, and Shiva Kasiviswanathan. Subsampled rényi differential privacy and analytical moments accountant, 2018.

[58] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially Private Generative Adversarial Network. *arXiv*, Feb 2018.

[59] Qingyu Zhao, Ehsan Adeli, Nicolas Honnorat, Tuo Leng, and Kilian M. Pohl. Variational AutoEncoder for Regression: Application to Brain Aging Analysis. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, pages 823–831. Springer, Cham, Switzerland, Oct 2019.

[60] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. Local Differential Privacy based Federated Learning for Internet of Things. *arXiv*, Apr 2020.