Titre: Title:	A Differential Private Approach for Preserving the Privacy in Deep Learning Algorithms
Auteur: Author:	Amir Barzegar
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Barzegar, A. (2021). A Differential Private Approach for Preserving the Privacy in Deep Learning Algorithms [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/6287/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/6287/
Directeurs de recherche: Advisors:	John Mullins, & Maleknaz Nayebi
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A differential private approach for preserving the privacy in deep learning algorithms

AMIR BARZEGAR
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

A differential private approach for preserving the privacy in deep learning algorithms

présenté par **Amir BARZEGAR**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Michel DESMARAIS, président
John MULLINS, membre et directeur de recherche
Maleknaz NAYEBI, membre et codirectrice de recherche
Giuliano ANTONIOL, membre

DEDICATION

To all my friends at the lab,
I will miss you...

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my research directors, Professor John Mullins and Professor Maleknaz Nayebi, for their guidance, supports, and the valuable words of advice during my graduate studies.

I would like to convey my gratitude to the jury members, Prof.Michel Desmarais and Prof.Giulio Antoniol for accepting to be a jury member.

I would like to say a big "thank you" to all my colleagues at the “CRAC Lab” who shared knowledge, wisdom, and joy with me.

Last but not least, I am deeply grateful to my family for their unconditional love and support.

RÉSUMÉ

L'apprentissage automatique a le potential de devenir une composante standard de chaque produit technologique et d'être utilisé par tous dans leur vie quotidienne dans un avenir proche. Bien que l'apprentissage machine nous offre de nouvelles capacités magnifiques, il a également des vulnérabilités qui peuvent être facilement exploitées par des attaquants.

Il devient de plus en plus nécessaire d'étudier les implications de l'apprentissage automatique sur la sécurité et la vie privée à mesure que l'apprentissage automatique est plus largement utilisé. Les aspects de l'apprentissage automatique liés à la sécurité ont fait l'objet de nombreuses recherches et d'une grande publicité, comme les attaques contradictoires, et les attaques fondées sur la protection de la vie privée ont reçu moins d'attention. Cependant, malgré le fait qu'il y a eu un nombre croissant de travaux dans ce domaine, il n'y a pas eu d'analyse approfondie des attaques liées à la vie privée. Les données d'entraînement et les modèles sont des éléments essentiels des systèmes d'apprentissage automatique. Nous devons protéger la vie privée de ces deux parties afin d'avoir un système d'apprentissage automatique qui soit protégé contre les atteintes à la vie privée. Dans les systèmes d'apprentissage automatique, il y a trois parties vulnérables qui rendent les données et le modèle accessibles. Dans un premier temps, nous avons les données de formation elles-mêmes, puisque les utilisateurs malveillants peuvent accéder aux données de formation pour récupérer des informations sensibles. La deuxième étape est le modèle lui-même, car si l'utilisateur malveillant accède au modèle, il pourra récupérer le modèle et les données de d'entraînement qui sont deux parties sensibles de l'apprentissage automatique. Le dernier élément vulnérable des systèmes d'apprentissage automatique est l'API de prédiction, car si l'attaquant y a accès, il pourrait récupérer le modèle et les données de formation. Pour contribuer à cet axe de recherche, nous avons proposé une architecture en plusieurs étapes pour les systèmes d'apprentissage automatique afin d'avoir un bon compromis entre la vie privée et la sécurité des systèmes d'apprentissage automatique de manière à préserver la vie privée des données de formation et du modèle dans trois parties vulnérables du pipeline d'apprentissage automatique (données de formation, modèle et API de prédiction). Comme contribution à cette ligne de recherche, nous avons proposé un modèle de formation selon la combinaison du moment adaptatif (Adam) avec une confidentialité différentielle que nous avons appelé DPAdam, pour préserver la vie privée du modèle de formation séparément.

Pour la partie implémentation, nous avons utilisé l'ensemble de données MNIST, un ensemble de données bien connu pour les tâches d'apprentissage automatique. Nous avons

implanté le réseau neuronal à convolution (CNN) avec diverses architectures (CNN simple à 3 couches, VGG-net et ALex-net) et DPAdam mesurer pour voir l'amélioration de notre recherche. Ainsi, dans le meilleur des cas, nous avons obtenu une précision de 97% pour l'entraînement avec une (7, 10^{-5})confidentialité différentielle, ce qui est acceptable dans des produits d'entreprise.

ABSTRACT

Machine learning has the potential and ability to become a standard component of every tech product and be used by anyone in their daily lives in the near future. Although machine learning provides us with some splendid new capabilities, it also has vulnerabilities that attackers can easily exploit.

It becomes more and more necessary to study the implications of machine learning on security and privacy as machine learning is more widely used. There has been much research and publicity on the security aspects of machine learning, such as adversarial attacks, and less attention has been given to privacy-based attacks, which means having unauthorized access to information.

However, despite the fact that there has been a growing body of work in this area, there has not been an extensive analysis of privacy-related attacks.

Both training data and models are crucial elements of machine learning systems. We should protect these two parts' privacy from having a machine learning system that is privacy protected. In machine learning systems, there are three vulnerable parts that make the data and the model accessible. In the first step, we have the training data itself since adversarial users can access the training data to retrieve sensitive information. The second step is the model itself. If the adversarial user accesses the model, they will retrieve model and training data, two sensitive parts of the machine learning system. The last vulnerable component of machine learning systems is the prediction API, and if the attackers have direct access to this part of system, they can retrieve the model and training data.

To contribute to this research line, we proposed a multi-step architecture for machine learning systems to a trade-off between privacy and accuracy of machine learning systems. We preserved the privacy of training data and model in three vulnerable parts of the machine learning pipeline (training data, model, and prediction API). The other contribution to this research line was that we proposed a training model according to the combination of the adaptive moment (Adam) with differential privacy. We called it DPAdam to preserve the privacy of the training model separately.

For the implementation part, we used the MNIST dataset as a well-known dataset for machine learning tasks for training and testing our model. We implemented the Convolution Neural Network(CNN) with different architecture (simple two-layer CNN, VGG-net, and Alex-net) and DPAdam to improve our training model's privacy in our research. Finally , in the

best case, we achieved an accuracy of 97% for training with (7, 10^{-5})-differential privacy for training algorithm. These two metrics are information leakage (privacy budget) and the probability of failure and and we want to keep the amount of information as much as possible near to zero.The amount of (7, 10^{-5})-differential privacy which is good privacy in enterprise products.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

DP	Differential Privacy
ML	Machine Learning
DNN	Deep Neural Network
CNN	Convolutional Neural Network
DB	Data Base
M	Mechanism
PCA	Principal Component Analysis
SGD	Stochastic Gradient Decent
Adam	Adaptive Moments
DPSGD	Differential Private Stochastic Gradient Decent
DPAdam	Differential Private Adaptive Moments
BP	Back Propagation
ReLU	Rectified Linear Unit
DILL	Data Illustration
dPA	deep private auto-encoder
Adgrad	Adaptive gradient
RMSprop	Root Mean Square prop
API	Application Programming Interface
RAPPOR	Randomized Aggregatable Privacy Preserving Ordinal Response
Conv	Convolution
FC	Fully Connected

CHAPTER 1 INTRODUCTION

Machine learning (ML) has grown in popularity due to extensive amounts of available data and hardware advances, both in academic research and real-world applications. While this is happening, machine learning is receiving greater attention for its impact on security, privacy. In terms of privacy, practically every online service collects our personal information and uses it to train models to power machine learning-based applications. As they are presented as black-box models, it is expected that they should not reveal any information about the data they were trained on that [7]. It is unpleasant if an attacker extract sensitive information, such as location, health records, or identity information, to train a model. Simultaneously, suppose third parties have used data without their consent. In that case, the same attack could be used to identify any unauthorized usage and thus work in the best interests of the users' privacy [7].

Privacy of users and their data and automation of the processes are two of the digital world's current concerns. These two areas have often being discussed orthogonally and independently from each other. ML systems are trained to build models and infer data patterns by looking at the raw and un-anonymized data. The private nature of this data, whether being medical record, financial transactions or user identity (Sweeney,1997 [8]; Alipanahi et al., 2015 [9]; Kannan et al., 2016 [10]; Kononenko, 2001 [11]) makes the ML tasks challenging.

In general, we need to move away from an accuracy-at-all-costs mindset toward an approach that is more balanced between accuracy and privacy, as shown in Figure 1.1 [2].

vs non robust.png vs non robust.png vs non robust.png vs non robust.png

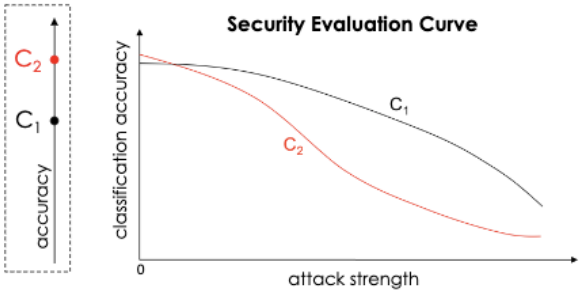


Figure 1.1 C1 and C2 are two models. C1 starts off being less accurate than C2, but it can also better withstand attack strength as it increases [2].

In data science and ML research, the main goal is to extract a pattern or some features after processing the data to produce a model for future predictions based on unseen data as a

general model. Sometimes, the model generalized well without over-fitting specific training examples, but some of these data are implicitly memorized and it is not cause revealing the information by the model. By applying ML in almost all fields of science, engineering, and human life, it is time to find a useful way to guarantee and preserve our sensitive data privacy. ML aims to extract the specific features from data, while privacy is safeguarded by hiding data from attackers and malicious activity [12] [13]. Differential privacy(DP) was first proposed as a privacy concept for data analyses. It relied on the notion of the definition of a population involves the learning of nothing about a specific individual while collecting useful information about that population as a whole [12]. We can propose ML algorithms that need to train models according to private information through the DP application. Learning with DP gives specific protection certifications and alleviating the danger of uncovering delicate preparing information in ML.

Imagine that we want to produce a global cancer prediction system, according to ML. Indeed, this system needs a primary dataset for the model's training and delivers an objective function that gives us the ability to predict cancer. The primary dataset contains much individual information like name, age, sex, genetic information, etc. If attackers reach these data, they may end up in possession of all critical data, including personal info on the health system or important data about a company that they want to exploit for data analysis. All these are critical in the concept of privacy. Using the DP for a designed system, privacy can be provided for the information and extract useful information that is beneficial for individuals and firms [14].

According to its pipeline, ML systems' vulnerabilities can be categorized into training datasets, training models, and prediction (production) phase vulnerabilities.

We explained the critical issue of privacy attacks on ML. According to it, our objective is to design an ML system to prevent privacy attacks. Our research questions are as below:

1. How to design an ML system to preserve privacy?
2. How to achieve an acceptable trade-off between privacy and accuracy in the ML system?
3. What are attack-able elements in the pipeline of ML?
4. How to preserve the privacy of the training dataset?
5. How to preserve the privacy of the training model and algorithm?
6. How to protect the privacy of predictions?
7. How to prevent the API attacks on ML systems?

8. How attackers gain access to ML systems by attacking them?

An ML security threat model is the answer to the above questions. It is a structured framework that defines the possible attack vectors on your ML system.

Confidentiality, integrity, and availability are three pillars of information security: the information security triad. If you can protect all three, you have got your security covered [15].

1. A confidentiality attack (Privacy attacks) aims to extract sensitive information from your machine-learning system. For instance, an attacker might want to infer whether a specific data point has been a part of a specific training dataset [16].

2. Integrity attacks aim to make your ML model make errors, but — importantly — do so invisibly. For example, an attacker might be attempting to place a malicious file as benign without affecting overall performance, so you never notice the error [17].

3. Availability attacks aim to take your ML system down completely. For instance, if enough lousy data is inserted into your training pool, the model's learning boundary becomes practically null and useless. It is a deniable of service (DOS) of the ML world [18].

If we categorize the security attacks, we can categorize them as below:

1. **Evasion attacks** (also called "adversarial examples")

   Generally, the exploitation that occurs at inference time is the most common type. It takes advantage of ML's inherent vulnerabilities [2].

2. **Poisoning attacks**

   During training time, both integrity and availability can be the goal. A malicious attacker could insert a few selected examples into your training set with the intent to build a backdoor into your model (integrity) or insert so much insufficient data that the model's boundary becomes useless (availability) [19].

3. **Privacy attacks**

   Privacy attacks may be the least researched. However, they are a significant threat today because attackers tend to extract private information from your ML model rather than interfere with its functionality [20].

Two targets of attackers in privacy attacks in ML systems are the training dataset and the learning model. First and foremost, the ML system's value comes from its data, so not just anyone can steal it [7].

If all you care about is recognizing cats in YouTube videos, data privacy probably isn't a big issue [21]. In other words, controlling the release of the training data with which an ML system is trained becomes a real problem once you begin applying ML to industries like banking and healthcare. According to how the machine is designed and trained, we may still be able to reverse engineer what the machine's memories witnessed during training. That can open the door to some privacy invasions.

A membership inference attack is the simplest type. It occurs when the attacker has a data point on hand and wants to know if it belonged to the original dataset. This technique, devised by Shokri et al. identified which patients could or could not be included in the hospital discharge dataset. According to queries that they send to the ML system's prediction API, they could build "shadow models" based on the original target model. On both the Google and Amazon "ML as a service" classifiers, the attackers proved the successful attack [22].

There is a second type of attack called data extraction or model inversion attacks, which attempts to extract an average representation of each of the classes on which the model was trained [23].

Besides the ML system itself, the model itself is also valuable, so you can see why people try to steal it (do "model extraction") [24]. First and foremost, your model is valuable because it is unique. Imagine your data science team has worked nonstop for six months to create a masterpiece that sets you apart from the competition.

The second is perhaps a bit less obvious but arguably riskier since they intend to attack you with a **White Box** evasion attack. The adversary has a much greater chance to breach your system if they can obtain your model's gradients before you [25].

The most critical problem of ML systems is privacy attacks because they can not preserve information privacy. Most of the time that attackers try to retrieve a training set from a produced model like the work of Fredrikson et al. [23] that they used hill-climbing on the output probabilities of a classifier to retrieve individual data from the training set. In the work of Shokri et al. [22] by using membership inference attacks against ML models, they were able to train a shadow model. Using this shadow model, they could predict if a data is in the training data set or not in the training dataset. In Papernot et al. [26] they tried to retrieve the training dataset by making black-box attacks on the ML model. To preserve information privacy, it appears challenging to accommodate using an ML mechanism to

secure information privacy. One of the primary ways for this preservation is anonymizing. The problem is that privacy can not be preserved by only data anonymization because most of these data are high-dimensional. Some of these data unique, which is identifiable even after anonymizing [27]. The Gaussian mechanism can protect information privacy and is one of the most reliable methods between DP models. This model can protect sensitive information privacy in the ML system, and incredibly Deep Neural Networks (DNNs) [28]. To overcome this defect, We decided to use the DP method based on the lack of anonymizing technique, uniquely using the Gaussian DP methods [12]. The defenses we can do for preventing our system from privacy attacks are as below:

1. **API privacy protection:**

   Most of the papers I cited above rely on having unrestricted access to the target model's API (with confidence scores), which makes it the easiest place to start, and by protecting this part of the system, we can have a more privacy-protected system

2. **Model privacy protection:**

   One of the most valuable elements of ML systems is the model, and by protecting the privacy of the model, we can lead to privacy preserved ML system.

3. **Dataset privacy protection:**

   The other valuable element f ML systems are the training dataset, and by preserving the privacy of the training dataset, we have a protected ML system.

DP is a theoretical framework that aims to provide a formal guarantee of robustness. It is more specifically meant to show that two models differing only by one sample are equivalent and produce similar predictions (making it impossible to infer that sample). In laymen's terms, DP is all about injecting noise (or "randomness") into your ML system. There are several ways to accomplish this: Change users' input into a common training pool (e.g., whenever a user sends data to a server, x% of it is replaced with random numbers). perturb the underlying data perturb the parameters of the model by injecting noise into the parameter update process Perturb the loss function (similar to, e.g., the L2 regularization) perturb output during prediction (e.g., send random noise to x%.) The challenge with DP is that it's not free. When you want to protect more, the more "privacy budget" you set aside, your model's worse performance suffers, and we should consider a trade-off between privacy and performance.

In the issue of security and privacy, there is no 100% secure system. It is almost impossible to have a 100% secure system, introduce a mechanism to reach a high level of security is

demanded. [29].

According to these gaps and defects of ML systems, we need a protected comprehensive system that preserves the training dataset's privacy, which is one of the vulnerable parts of ML algorithms. Second, protecting the learning algorithm's privacy with our proposed training algorithm, which we call it DPAdam. Finally, the other problem of the core base system is that the attacker can directly attack systems by being on the edge of attack. However, by putting the central decision core in the back-end and use the other layer of training, as we called it understudy, we prevent API attacks, which is the other famous attack. The understudy element in our proposed architecture is shown in 1.2.

In this work, we used a MNIST [30] dataset as the input and protected the privacy of training and test set by adding calibrated noise according to DP as a first phase. By having this phase, we protect our information with DP methods. In this way, we follow up for some reasons; First, as the data are privacy protected, if an adversarial user accesses our data, they can not reveal the sensitive information. Second, as we add noise to our data to preserve privacy, this preservation will prevent the model from memorizing our training set and control the model from over-fitting. Third, when working with a big dataset as an input of the Deep neural network (DNN), when we work with a big dataset, if we perturb the training set with a controlled amount of ε and σ, the two elements of the DP concept, we can guarantee that our system is (ε, δ)-DP. In this way, we have protected our information's privacy and not ruined the system's performance. We can set a good trade-off between security and performance. The second phase of protecting the privacy of information is using the structured knowledge aggregation technique, which has been transferred gradually from 1994 to 2016 (Breiman, 1994 [31], Nissim et al., 2007 [32], Pathak et al., 2010 [33] and Hamm et al., 2016 [34]) and using a group of separated DNN for a pre-processed dataset with no overlapping between each other which we called Instructor. In the third phase, we use the next layer of privacy, called the understudy part. The understudy part is trained on the aggregated output of the previous instructor group. Finally, in this way, we can make sure that the understudy training does not depend directly on sensitive information on the training set, and this phase will prevent API attacks [24]. We have proposed a method based on DP in a multi-layer architecture system because of using the composition feature of DP [12] we will increase the privacy of information. In each layer, we preserve the privacy of data, and finally, the whole system would be protected, and it is more difficult for the attacker to penetrate this system.

In the work of Abadi et al. [13], they work on the privacy of learning algorithm they limited the number of accesses between the first and second levels because the second level of privacy could expose the knowledge of the first level as there is an interaction between levels. In our

work, we try to improve the privacy of the training dataset and training algorithm. Adding a new phase in our architecture, as we call it the "understudy," we can learn from "Instructor" without limitation.

Finally, we used the MNIST [30] benchmark to evaluate our work. In this work, we can limit DP metrics to a range between (0.0, 10), the same as real-world privacy protection mechanisms like Google RAPPOR introduced by Erlingsson et al. [35].

Our work's contribution is to preserve information privacy in the raw training data and learning algorithm. Suppose the attacker reaches our training data. In that case, they do not reveal the training set's information, and if they can penetrate our algorithm, they do not be able to extract the model from our learning model.

First, we preserve the privacy of training data by using DP. We add a calibrated amount of noise according to the dataset's sensitivity to protect the training dataset's privacy. Second, by introducing a new training model that combines the DP and Adam training model and called it DPADAM, we preserve ML models' privacy. Finally, for protecting our prediction(production) privacy, we used a second layer on training, as we called the "understudy" layer, which will prevent the API attacks on the core of learning models. A general overview of our system is shown in Figure 1.2.

arch.png arch.png arch.png arch.png

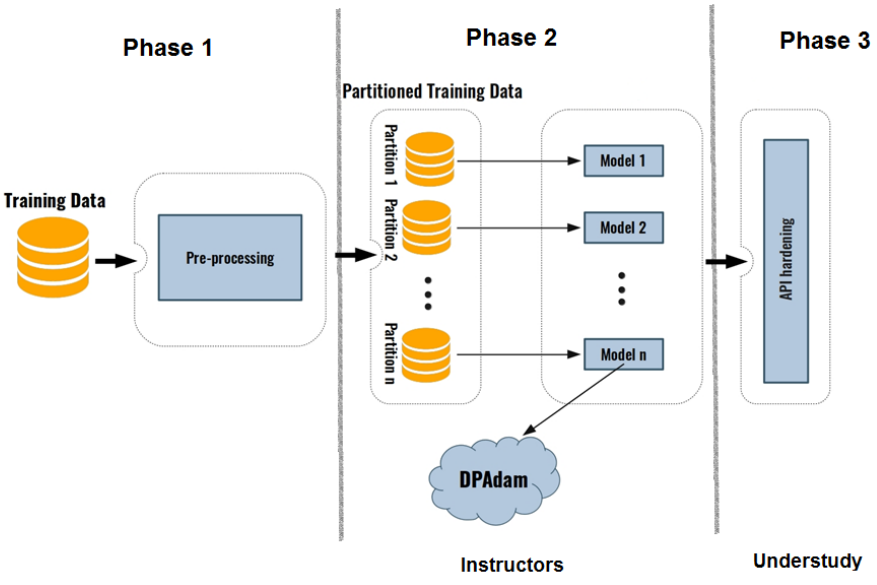


Figure 1.2 A general view of our work.

To over the gaps, as we mentioned, we proposed two contribution in our work as below:

1. Proposing a multi-step privacy preserving architecture:

   By proposing a multi-step architecture for preserving the privacy of data in three different steps.

2. Proposing a privacy protected learning model:

   By combining the Adam learning model with differential privacy.

In the second chapter, we will review the background knowledge. In chapter 3, we have discussed the methodology. In the 4th chapter , we will discuss the results. Finally, the last chapter would be our conclusion.

CHAPTER 2 PRELIMINARIES

In this section, we intend to introduce the fundamentals of DP and ML and expand their features, which we use in our system. We speak about our mechanism and the straightforward way to reach privacy and exemplary performance in ML systems in the following parts.

2.1 Related Works

Researchers have covered the problem of preserving data privacy and information in ML or data mining algorithms since the 90s [13]. We can classify the related work to the privacy of training data, the privacy of the training model and algorithm, and the prediction layer's privacy. These three parts are crucial elements of an ML system. If we can protect these three parts, we can have a total protected system [7].

The first works on privacy-preserving learning were done as a secure function evaluation and multi-part computation framework. In these works, researchers split the input into multiple partitions and attempt to minimize information loss during mutual communication. In our work to increase the ML system's privacy, we also use secure multi-party computation [36] to reduce the information loss and increase privacy. In this case, if the attacker can penetrate our model, they would be able to attack one of our n separate models, and the rest of the models and training algorithms would be secure.

One approach uses k-anonymity, which entails generalizing and suppressing specific identifying characteristics, which can provide a significant level of protection to both the principal and particular characteristics [37]. Nevertheless, these methods have substantial limitations, both theoretically and empirically [27, 38], and this protection is weak against linkage attacks [39]. We keep the primary raw records protected and do computation and learning on a protected training set. The reason for learning with a protected training set is that to preserve the training set from model inversion attacks [23]. For sure, training the model with a protected dataset will decrease the ML system's performance, but we should set a good trade-off between privacy and performance.

The DP theory provides the investigative framework for our work. It has been applied to an extensive collection of ML tasks distinguished from the training dataset, the training mechanism, and the prediction layer.

The work of Bun et al. [40] introduces a relaxation of DP (generalizing the work of Dwork and Rothblum [28].) All in all, these works illustrate that the moment's accountant is a helpful

technique for theoretical and empirical analyses of complex privacy-preserving algorithms [13].

A typical target for learning with privacy is convex optimization problems bowed to various techniques in learning algorithms. In the work of Wu et al. [41], they tried to increase the accuracy of the ML system by introducing the private SGD. They also tried to decrease the run time ML algorithm because of the overhead of adding noise to the system [41], and they achieve 83% accuracy on MNIST via convex empirical risk minimization [13].

The research of McSherry and Mironov [42] introduce the differential private recommender systems. This work was the first end-to-end differentially private system evaluated on the Netflix Prize dataset [13]. Their work consists of two parts. The first part is a differential private aggregation learning phase, and the second part is an individual recommendation phase. However, the problem shared many similarities with our model class in high-dimensional inputs and non-convex objective function. They identified the core of the learning task, virtually adequate statistics, computed in a differentially private method via a Gaussian mechanism. In our approach, no such sufficient statistics exist.

In the research of Shokri and Shmatikov [43], they designed and assessed a system for distributed training of a deep neural network with multiple parties to jointly learn a neural network model without sharing their training datasets [43]. Participants, who hold their data near, communicate sanitized updates to a central authority. The sanitization relies on an additive-noise mechanism based on a sensitivity estimate, which could be improved to a demanding sensitivity guarantee. They compute privacy loss per parameter, and they don't care about the whole model's privacy [13].

The recent approach towards differentially private deep learning is explored by Phan et al. [44]. This work is related to auto-encoders, and they proposed the method of deep private auto-encoder (dPA). In their work, they reach ε-DP by perturbing the autoencoder's objective functions of traditional deep auto-encoder [44].

Abedi et al. [13] proposed a differential private learning algorithm according to the combination of Stochastic gradient descent (SGD) learning technique and DP. In comparison to our work, according to the disadvantages of SGD, as it may stick in local minima and its low convergence speed, we decided to use the Adam learning algorithm. In our work, we have proposed an architecture for preserving ML systems' privacy in three-phase. In the first phase, we maintain the training set's confidentiality as a crucial element of ML systems by our preprocessing. In the second phase, we introduced the other learning algorithm according to DP by using Adaptive moments (Adam) [45] and DP to overcome the SGD learning model's defects. We used the ensemble technique to increase the privacy of the model. In

the third phase, by adding a new training model to our architecture to make the two first phases a back box system. We use the second phase results and use the third phase as the whole system's prediction layer to increase the total system's privacy. Because most of the time, attackers try to reach training datasets and learning algorithms by using the prediction layer of an ML system.

In the recent work of Phan et al. [46], they propose adaptive Laplacian mechanism as they call it AdLM. In this mechanism, they add laplacian noise into the feature selection phase and, according to each feature's importance, add less or more noise to their data and protect the privacy of training set data. They do not care about their model and also the prediction layer. We treat our training dataset the same in our research, and we protect the raw dataset, not only features. We cover the privacy of the model and prediction layer, too [46].

2.2 Differential Privacy

The problem of preserving privacy through data analysis spans different disciplines over a long period. Electronic data about individuals become ever more detailed, and technology enables ever more powerful means of collecting and curating this data. We see an increase in the need to define privacy robustly, mathematically rigorously and provide computationally sophisticated algorithms that satisfy this definition. DP is such a definition. DP is a security definition initially created by Dwork et al. throughout the years [12].

DP is a framework for openly sharing data about a dataset by showing the examples of gatherings in the dataset while preserving the data regarding the entries in the dataset.

DP provides privacy by the process, especially the introduction of randomness. One of the first examples of privacy by a randomized process is a randomized response, a response system devised in the social sciences to gather statistical data about embarrassing or illegal behavior captured by having a property P. By using randomness, we can have plausible deniability [12]. In detail for showing the property of P we will follow the below structure:

1. Flip a coin.
2. If *tail*, then response truthfully.
3. If *head*, flip a second coin and:
   (a) If *head*, respond "Yes."
   (b) If *tail*, respond "No."

Privacy comes from the plausible deniability of any possible outcome; in particular, if owning property P corresponded to engaging in illegal behavior, even an answer of "Yes" is not incriminating because the probability of receiving this answer is at least $\frac{1}{4}$ regardless of whether or not the respondent owns property P. Accuracy comes from an understanding of the noise generation process (which is introducing spurious "Yes" and "No" responses during randomization): Answering "Yes" is $\frac{1}{4}$ times the number of participants who do not have property P plus $\frac{3}{4}$ times the number who have property P. Thus, if p is the true fraction of participants having property p, the expected number of "Yes" answers is $(\frac{1}{4})$(1-p)+$(\frac{3}{4})$p = $(\frac{1}{4})+\frac{p}{2}$. As a result, we can estimate p as twice the fraction answering "yes" minus $\frac{1}{2}$; that is, $2((\frac{1}{4}) + \frac{p}{2})\ \frac{1}{2}$. More precisely, randomization is essential. Any non-trivial privacy guarantee that holds regardless of all present or even future sources of auxiliary information, including other databases, studies, web sites, online communities, gossip, newspapers, government statistics, requires randomization.

The critical issue here is defining the input and output space. In general, a randomized algorithm with domain A and range B will be mapped from A to the probability simplex over discrete B, which we show by $\Delta(B)$:

Definition 1. *(Probability Simplex). When we have discrete set B, the probability simplex over B would be as below:*

$$\Delta(B) = \left\{ x \in \mathbb{R}^{|B|} : x_i \geq 0 \text{ for all } i \text{ and} \sum_{i=1}^{B} x_i = 1 \right\}$$

Definition 2. *(Randomized Algorithm). A randomized algorithm M with domain A and discrete range B is associated with a mapping:*

$$M : A \longrightarrow \Delta(B)$$

.

On input $a \in A$, the algorithm M outputs M(a) = b with probability $(M(a))_b$ for each $b \in B$. The probability space is over the coin flips of the algorithm M [12].

Lets a database x contain a collection of records from a universe χ and we can represent databases by their histograms : $x \in N^{|\chi|}$, in which each entry x_i represent the number of elements in the database x of type $i \in \chi$. Here N is the set of all non-negative integer including zero. So according to this description the distance between two databases x and y will be their norm1 distance:

Definition 3. *(Distance between databases). The norm1 of a database x is denoted by $||x||_1$ and calculated as below:*

$$||x||_1 = \sum_{i=1}^{\chi} |x_i| \tag{2.1}$$

So the distance between two databases x and y is $||x - y||_1$.

$||x-y||_1$ will show how many records differ between x and y (the database records are binary).

Now we can define the DP, which guarantees that a randomized algorithm behaves the same on a similar database as input.

Definition 4. *(Differential Privacy). A randomised algorithm M with domain of $N^{|\chi|}$ is (ε, δ)-differential private if for all $S \subseteq R$ (R is the range of M algorithm)and for all $x, y \in N^{|\chi|}$ such that $||x - y||_1 \leq 1$:*

$$\Pr[M(x) \in S] \leq exp(\varepsilon) \Pr[M(y) \in S] + \delta$$

Where the probability space is over the coin flip (a randomized mechanism) of algorithm M [12].

Another approach to depicting DP is a limitation on the calculations used to distribute total data about a measurable database that restricts private data of records whose data is in the database. DP is a standout amongst the most well-known meanings of protection today.

The accompanying subsections scientifically characterize differential protection and present some commonly utilized techniques in DP. One kind of DP is ε-DP, a mathematical definition of privacy loss related to any data.

2.2.1 ε -Differential Privacy

A randomised algorithm M with domain of $N^{|\chi|}$ is (ε)-differential private if for all $S \subseteq R$ (R is the range of M algorithm) and for all $x, y \in N^{|\chi|}$ such that $||x - y||_1 \leq 1$:

$$\Pr[M(x) \in S] \leq exp(\varepsilon) \Pr[M(y) \in S]$$

If the mechanism has these features, we call it as ε -DP.

Generally we denote the query by $f : X \to Y$ and the effect of mechanism M over this query would be like :

$$M(x) = f(x) + noise \tag{2.2}$$

A mechanism is called ε-DP if it satisfy the following condition :

for all $x, y \in X$ with $d(x, y) = 1$, and for all measurable set $S \subset R^n$ (R is the range of M and S is the subset.)

$$\Pr[M(x) \in S] \leq e^{\varepsilon} \Pr[M(y) \in S] \tag{2.3}$$

This implies given the outcomes from anonymous questions on two available databases that contrast by one line. It is difficult to figure out which result is from which database. Laplace mechanism is one of these methods, which is ε-DP [12]

2.2.2 The Laplace Mechanism

Numeric queries function $f : N^{\chi} \to \mathbb{R}^k$, are one of the most fundamental types of databases queries. These queries map databases to k real number. One of the most important parameter to show the accuracy of our answer to our queries is their ℓ_1 sensitivity:

Definition 5. *($\ell_1 -$ sensitivity). The ℓ_1-sensitivity of a function $f : N^{|\chi|} \to \mathbb{R}^k$ is:*

$$\Delta f = \max_{x,y \in N^{|\chi|}, ||x-y||_1 = 1} ||f(x) - f(y)||_1 \tag{2.4}$$

Definition 6. *($\ell_2 -$ sensitivity). The ℓ_2-sensitivity of a function $f : N^{|\chi|} \to \mathbb{R}^k$ is:*

$$\Delta_2 f = \max_{x,y \in N^{|\chi|}, ||x-y||_1 = 1} ||f(x) - f(y)||_2 \tag{2.5}$$

A function's sensitivity gives an upper bound on how much we must perturb its output to preserve privacy. It is a limitation for the amount of noise that we add to our data to maintain privacy.

The difference between $L1$-norm and $L2$-norm is that the $L2$-norm is less robust to outliers. In the formula of $L2$-norm, because of power of 2 in the procedure, it will increase the cost of outliers exponentially, but $L1$-norm increase linearly and is more robust, on the other hand, $L2$-norm is more stable than $L1$-norm and also the computational cost of that is higher than $L1$-norm [47, 48].

There is a different definition for counting the amount of estimation, [28] here is the algorithm

for the sensitivity of function with Laplace distribution:

Algorithm 1: Sensitivity with Laplace distribution Algorithm

Result: F(X) + L(Noise)
Initialization: Function F, Input X, Privacy level ε;
Compute F(x);
Compute sensitivity of F(X);
Compute the amount of noise based on Laplace distribution;
return F(X)+L;

In this algorithm, if the number of sensitivity increases, we can add more noise to data while preserving privacy and utility. While counting the algorithm, the amount of sensitivity was fixed to 1, and the amount of noise was constant, but here we can change the amount of noise based on this equation:

$$Noise = 2 * \left(\frac{S}{\epsilon}\right)^2 \tag{2.6}$$

If we choose the smaller value of ε, it means the amount of noise will increase, and privacy increases too [12].

Definition 7. *(Laplace distribution) The Laplace Distribution (centered at* 0*) with scale b is the distribution with probability density function with* $b > 0$ *[12]:*

$$Laplace(x|b) = \frac{1}{2b} e^{(-\frac{|x|}{b})} \tag{2.7}$$

The variance of this distribution is $\sigma^2 = 2b^2$. In some cases, we write Lap(b) to denote the Laplace distribution with scale b. Now we define the Laplace mechanism:

Definition 8. *(The Laplace Mechanism). Given any function* $f : N^{|\chi|} \to \mathbb{R}^k$*, the Laplace mechanism is defined as [12]:*

$$M_L(x, f(.), \varepsilon) = f(x) + (Y_1, ..., Y_k) \tag{2.8}$$

where Y_i *are random variable which we get from* $Lap(\frac{\Delta f}{\varepsilon})$ *[12].*

Theorem 1. *The Laplace mechanism preserves* $(\varepsilon, 0)$*-DP [12].*

Here, as we want to produce the amount of noise with Laplace distribution , we replace the noise with Lap(b) and we have:

$$M(x) = f(x) + Lap(b) \tag{2.9}$$

and the Laplace mechanism is ε -DP if :

Let $x \in N^{|\chi|}$ and $y \in N^{|\chi|}$ be such that $||x-y||_1 \leq 1$, and let $f(.)$ be some function $f : N^{\chi} \to \mathbb{R}^k$. $\Pr_x$ denote the probability density function of $M_L(x, f, \varepsilon)$, and $\Pr_y$ denote the probability density function of $M_L(y, f, \varepsilon)$. We compare the two at some arbitrary point $z \in \mathbb{R}^k$ [12]:

$$\frac{\Pr_x(z)}{\Pr_y(z)} = \prod_{i=1}^{k} \left(\frac{\exp(-\frac{\varepsilon|f(x)_i - Z_i|}{\Delta f})}{\exp(-\frac{\varepsilon|f(y)_i - Z_i|}{\Delta f})} = \exp\left(\frac{\varepsilon||f(x)_i - f(y)_i||_1}{\Delta f}\right) \leq exp(\varepsilon)\right.$$

where the first inequality follows from the triangle inequality, and the last follows from the definition of sensitivity and the fact that $||x-y||_1 \leq 1$ [12].
We can not use ε-DP for the most commonly used noise. Because based on using that definition, it is more specific with lower computational overload, but for generalizing DP, we need to relax its definition, so we have (ε, δ)-DP as bellow [12]:

Definition 9. *A mechanism M is (ε, δ)-Differential Private if for all x, $x^{'} \in X$ with d(x, $x^{'}$) = 1 and for all measurable $S \subset R^d$ and $\delta < 0$ [12]:*

$$\Pr\left(M(x) \in S\right) \leq e^{\epsilon} P(M(x^{'}) \in S) + \delta \tag{2.10}$$

One of the alternatives for Laplacian noise is Gaussian noise.Let $f : N^{|\chi|} \to \mathbb{R}^d$ be a d-dimensional function, and define its ℓ_2 sensitivity to be $\Delta_2 f = \max_{x,y} ||f(x - f(y))||_2$. The Gaussian mechanism with the parameter σ add noise scaled to $N(0, \sigma^2)$ to each of d component of output [12].

Definition 10. *Gaussian mechanism, given a query $f : X \to Y$, Gaussian mechanism M adds a Gaussian noise to the query [12]:*

$$M(x) = f(x) + N(0, \sigma^2 I) \tag{2.11}$$

Theorem 2. *With arbitrary amount of $\varepsilon \in (0, 1)$ and for $c^2 > 2\log\frac{1.25}{\delta}$, the Gaussian mechanism with parameter $\sigma \geq \frac{c\Delta_2 f}{\varepsilon}$ is (ε, δ)-differentially private [12].*

Then the Gaussian mechanism is (ε, δ)-DP provided if it satisfy the below formula [12]:

$$\varepsilon = \sqrt{2log(\frac{1.25}{\delta})} \times \frac{\Delta_2 f}{\sigma} \tag{2.12}$$

Gaussian noise offers several advantages, including the fact that the same type of noise

that we add for privacy is like the other source of the noise; besides, since the sum of two Gaussian distributions is itself a Gaussian distribution, researchers may have an easier time understanding how the privacy mechanism affects statistical analysis [12].

The two Gaussian mechanisms lead to the same cumulative loss as the composition theorem. Even though the privacy guarantee is weaker for each computation, many computations' cumulative effects are comparable. The other benefit is that if δ is sufficiently small, we will never experience the guarantee's weakness in practice.

How to choose δ ? The δ is also known as the probability that something goes wrong. A standard option is to pick δ that is significantly smaller than $\frac{1}{n}$, where n is the total number of entries in the dataset. The reasoning is as follows: there is a possibility of δ that each individual's data will leak in the worst possible case scenario. Therefore, the total odds that someone's data leaks are $\approx n\delta$: we need to make sure that this number is small enough [12].

How to choose ε ? The ε in ε-DP is the maximum amount of knowledge gained by the attacker. We can describe this knowledge gain in Bayesian terms, where the attacker is trying to guess if the actual database D is D_1 or D_2. According to the definition of DP, We observe that ε bounds the betting odds' evolution. For each O, we have [12]:

$$\frac{\Pr[D = D_1 | M(D) = O]}{\Pr[D = D_2 | M(D) = O]} \leq e^{\varepsilon} . \frac{\Pr[D = D_1]}{\Pr[D = D_2]}$$

We can also calculate the quantity of a given output O. Let's define [12]:

$$\mathcal{L}_{D_1,D_2}(O) = \ln \frac{\frac{\Pr[D=D_1|M(D)=O]}{\Pr[D=D_2|M(D)=O]}}{\frac{\Pr[D=D_1]}{\Pr[D=D_2]}}$$

The denominator corresponds to the initial odds forD_1 verses D_2. How likely are one option versus the other before looking at the result of the mechanism? In Bayesian terms, this is known as the "prior." Meanwhile, the numerator of the fraction represents the probability of success. DP guarantees that $\mathcal{L}_{D_1,D_2}(O) \leq \varepsilon$ for all O [12].

Bayes' rule allows us to reformulate this quantity

$$\mathcal{L}_{D_1,D_2}(O) = \ln \left(\frac{\Pr[M(D_1) = O]}{\Pr[M(D_2) = O]} \right)$$

This is called the privacy loss random variable. The reason for being random variable is that

we consider $\mathcal{L}_{D_1,D_2}(O)$ when O changes according to $M(D_1)$, that we assume is the actual dataset.

An example In the context of DP, the normal distribution is called "Gaussian noise". Consider the following example: we're counting how many people in the dataset have blue eyes. We make this private by adding Gaussian noise, of variance $\sigma^2 = 2$ The attacker hesitates between two datasets: one with 1000 blue-eyed people, the other with 1001. The real number is 1000, but the attacker does not know that. The two distributions appear as follows in figure 2.1:

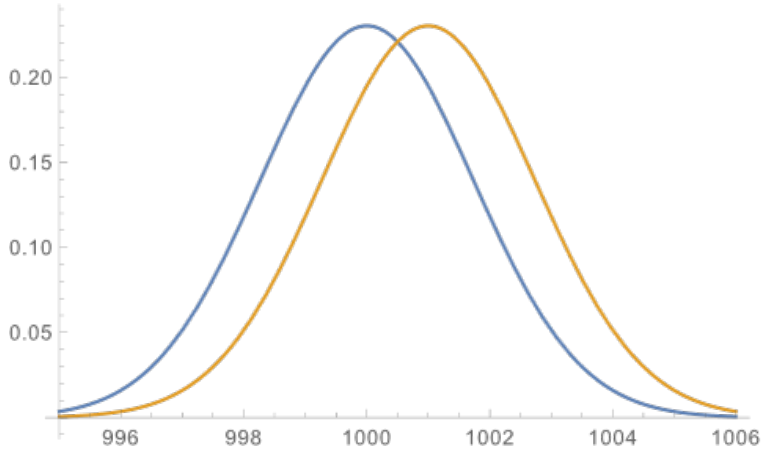


Figure 2.1 Adding Gaussian noise, of variance $\sigma^2 = 2$. [3]

It looks reasonable, so let's see what $e^{\mathcal{L}}$ looks like figure 2.2:

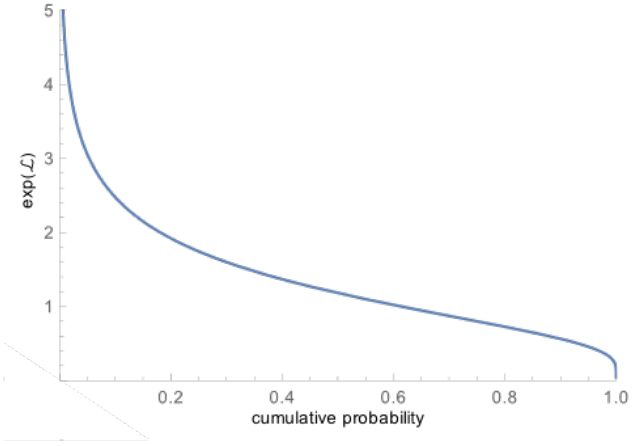


Figure 2.2 $e^{\mathcal{L}}$ of dataset distribution [3].

Look at the left line that reaches infinity. We can't just draw the line at e^{ε} and say "everything is underneath." What do we do? Here we use a δ to cover this gap.

We said that the δ in (ε, δ)-DP is the probability that something terrible happens. What does that mean in the context of Gaussian noise? First, we pick an arbitrary ε, say, ε=ln(3). Then, we look at how likely it for $e^{\mathcal{L}}$ to be above the e^{ε}=3 line.

It's not difficult to do: the X-axis is the probability space, so we measure the bad events' width as we can see in figure 2.3.

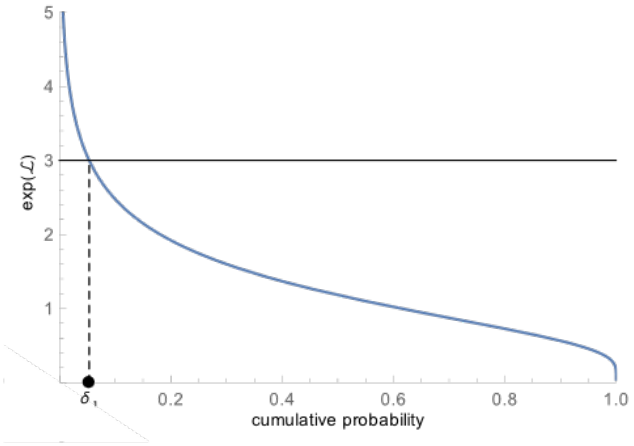


Figure 2.3 Measuring the bad events for defining the amount of δ.

It is logical that this would be so: this mechanism is $(ln(3), \delta_1)$-DP, with $\delta_1 = \approx 0.054$ [3].

2.2.3 Exponential Differential Privacy

The exponential mechanism is a mechanism for designing differentially private algorithms. It was developed by McSherry et al [12] [49].

The exponential mechanism is designed for a condition when we want the best response, but the amount of added noise to our data will destroy its value. In this mechanism, we have utility function maps the output pairs to utility score. The sensitivity of the utility score will be counted based on the range of a function(R) as below:

$$\Delta u = \max_{r \in R} \max_{||x-y||_1 \leq 1} |u(x,r) - u(y,r)| \tag{2.13}$$

The exponential mechanism ε takes in the scoring function score, and a data set x and parameter ϵ and does the following:

Definition 11. *(The Exponential Mechanism). The exponential mechanism* $M_E(x, u, R)$ *selects and outputs element* $r \in R$ *with probability proportional to* $\exp(\frac{\varepsilon u(x,r)}{2\Delta u})$.

Most of the initial research in DP revolves around real-valued functions with low sensitivity to change in a single individual's data and whose usefulness is not prevented by small additive perturbations. The exponential mechanism helps to extend the notion of DP to address these issues. Moreover, it describes a class of mechanisms that includes all feasible differentially private mechanisms [12].

2.2.4 Noisy Counting

Imagine that we have a simple database for credit rating as below:

Credit rating	
Credit rating	Count
Bad	3
Normal	1510
Good	200

Table 2.1 A sample of credit card rating dataset [1].

An attacker wants to know the number of bad credit customers. Instead of returning the actual amount of data, we return the ground truth with added random noise.

In this model, we have some data, which are our real data, and we call each one of them as N, then we choose a random number L from a Laplace Distribution, which its mean is located in zero. Its standard deviation is 2, and finally, we return $N + L$ [4].

In this way, when we add noise to our data, the data analyzers would analyze the data and gain some utility from our database. On the other hand, our sensitive data is secure, but this model has some advantages and disadvantages.

Simulated responses for the data in the credit rating database for counting the number of bad credit customer with the protected model would be as below:

Simulated responses for bad credits number	
Query number	Responses
1	2.91
2	1.88
3	1.29
4	4.02
5	5.34
Average	3.09
90% confidence interval	[1.69, 4.48]

Table 2.2 Simulated responses for the data in the credit rating database for counting the number of bad credit customer with the protected model [1].

For advantages, simultaneously, we have utility and privacy. Still, after several attempts of adversarial users, they would be able to reach our dataset's primary value, which means they would not be able to go to data at first. They can get closer to actual values after more attempts, but not the exact ones [1]. For example, in our simulation, the attacker will make a decent estimate after 50 queries, as shown in figure 2.4.

When we perturb the actual value of data by adding a big amount of noise, the attacker would not be able to reveal the real value of data. On the other side, when we add a large amount of noise to our data, it will change them to meaningless data. For example, imagine that we have some data that adds a large amount of noise, far from the real value. In this case, we have preserved the privacy of information. Still, in reality, we can not use the valuable statistical feature of these data, and they do not reflect a close feature of real data with preserved privacy. All the time, we should set a trade-off between security and utility, and they call this sensitivity, and the amount of noise we can add to our function is called the sensitivity of function [50] [49].

2.2.5 Privacy Budget

When we design a differentially private system, we set an upper bound limitation on the amount of difference anyone's participation can make. This upper-determined limitation is

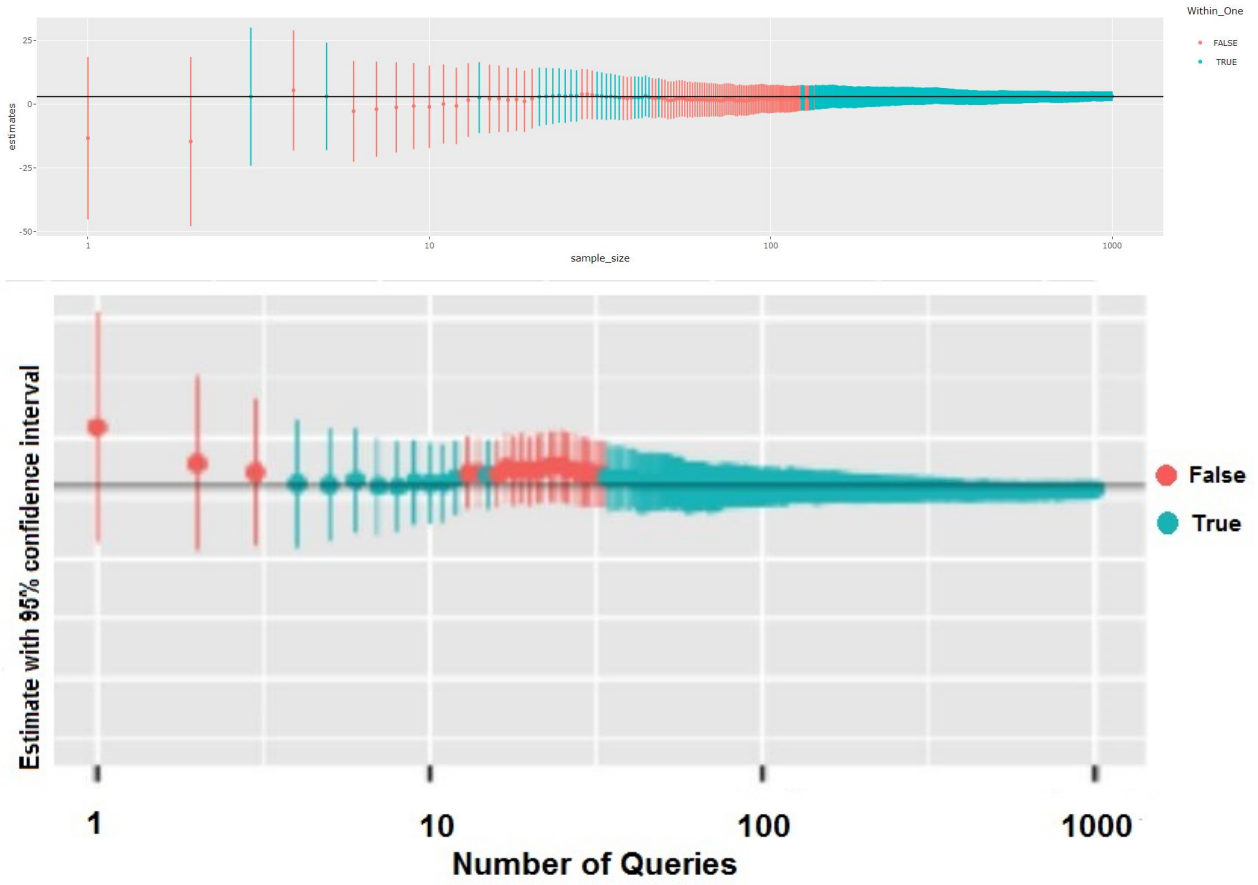


Figure 2.4 Confidence intervals of simulated noisy counting mechanism over 1000 queries and according to the results the attacker will make a decent estimate after 50 queries [4]

called the privacy budget. The system will allow one person to ask one question or ask a series of questions whose total impact on final output is no more than that budget. To make it more understandable, think of a privacy budget the same as money. Maybe your privacy budget is $1.00. You could ask a single $1 question if you would like, but you could not ask any more questions. Alternatively, you could ask one $0.30 question and seven $0.10 questions.

Some metaphors are dangerous, but comparing cumulative privacy impact to a financial budget is a good one. You have a total amount you can spend to reach your limitation, and you can choose how you spend it.

The only problem with privacy budgets is that they tend to be overly cautious and count as the worst-case because they are based on worst-case estimates. There are several ways to relieve this. A simple way to stretch privacy budgets is to cache query results. If you ask one

question two times, you get the same result both times, and you are only charged once [51].

Suppose we set the amount of this privacy budget with a high upper bound limitation. In that case, our system's privacy decreases because the attacker can spend this budget, asking many questions from our system to get feedback, but we can not set it to zero. Because our system should answer the queries, there should be a trade-off between them [52, 53].

2.2.6 Composition Theorems

So far, we have perceived how a system made of a single inquiry and noise can be demonstrated to be differentially private. Privacy is essential when you make a few components, combinatorially or successively [12]. For composition theorem, we can have Independent Composition and Adaptive Composition

Definition 12. ***Independent Composition*** *When we have K separate mechanism with independent noise, it would be Independent Composition, which we denote as* $M = (M_1, M_2, ..., M_k)$

Definition 13. *When we have K separate mechanism with independent noise and each mechanism has different or same number of parameters in Y where Y is* $(M_i(y_{1:i-1}))$ *and the output of each mechanism in defined as bellow [12]:*

$$x_{i_1} = M_1(x) \tag{2.14}$$

$$x_{i_i} = M_i(\xi_1, \xi_2, ..., \xi_{i-1})(x) \tag{2.15}$$

The Adaptive Composition of $M_{1:k}$ would be

$$M(x) = (\xi_1, \xi_2, ..., \xi_k) \tag{2.16}$$

At first, we have an elemental composition theorem for (ε, δ)-Differential Private with adaptive mechanisms [12].

Theorem 3. ***Basic Composition Theorem*** *Having k mechanism* $M_{1:k}$ *where each mechanism is* $(\varepsilon_i, \delta_i)$*-Differential Private with independent noise, the adaptive composition of* $M_{1:k}$ *is* $(\sum \varepsilon_i, \sum \delta_i)$*- Differential Private [12]*

As the up definition is for an underlying condition, in a different situation, the description also improves, and we have an Advanced composition theorem for(ε, δ)-DP with independent mechanisms.

Having k mechanism $M_1, M_2, ..., M_k$ and each one of them is ε-Differential Private, then the composition of these k mechanisms is:

Theorem 4. ***Advanced independent Composition Theorem***

For all $\varepsilon, \delta, \delta^{'}$, the class of (ε, δ)-differential private mechanism satisfies $(\varepsilon^{'}, k\delta+\delta^{'})$-differential private under k-fold adaptive composition for [12]:

$$\varepsilon^{'} = k(\varepsilon)(e^{\varepsilon} - 1) + \sqrt{2k \log \frac{1}{\delta'}\varepsilon} \tag{2.17}$$

One of the best and helpful corollary guide us to a safe choice of ε for each of k mechanism. If we want to guarantee $(\varepsilon^{'}, k\delta + \delta^{'})$-DP for a given $\varepsilon^{'}, \delta^{'}$.

Corollary 5. *Given target privacy parameter $0 < \varepsilon^{'} < 1$ and $\delta^{'} > 0$, to guarantee $(\varepsilon^{'}, k\delta + \delta^{'})$ cumulative privacy loss over k mechanism, it suffices that each mechanism is (ε, δ)-differentially private where [12]:*

$$\varepsilon = \frac{\varepsilon^{'}}{2\sqrt{2k \log \frac{1}{\delta'}}} \tag{2.18}$$

This corollary gives a rough guide for setting ε to get desired privacy parameters under composition [12].

2.3 Annonymization

Anonymization is a helpful privacy model to protect data in specific sharing scenarios (e.g., within an organization or with trusted partners).

Anonymizing direct identifiers is not enough if individuals may be re-identified through quasi-identifiers (such as postal code, date of birth, etc.) when linked to other available data sets. K-anonymity ensures that you cannot identify a single individual from a data set, no matter what additional information you hold about them [12].

Achieving k-anonymity needs to be at least k other individuals in the data set that share the same identifying attributes. In this case, k-anonymity can be achieved by generalizing the data.

2.3.1 k-anonymity

K-anonymity is a property possessed by specific, anonymized data. There are two common methods for achieving k-anonymity for some value of k [54].

1. **Suppression** : In this method, certain values of the attributes are replaced by an asterisk "*." All or some values of a column may be replaced by "*" [54].

2. **Generalization**: In this method, individual values of attributes are replaced by a broader category [54].

2.3.2 Linkage Attack

By combining anonymized data with background information, linkage attacks aim to re-identify individuals. The "linking" uses quasi-identifiers, such as zip or postcode, gender, salary, etc., present in both sets, to establish connections [55].

Many organizations are not aware of the linkage risk involving quasi-identifiers. While they may mask direct identifiers, they often do not think of hiding or generalizing the quasi-identifiers.

Let us provide an example; imagine that a healthcare provider shares anonymized data with external researchers about medical conditions. The export contains "Postal code," "Date of birth," "Gender," and "Description." An attacker could easily use a general voter list that contains "Name," "Gender," "Postal code," and "Date of birth" to cross-reference the patients.

The more you preserve the dataset's analytical features, like "Gender" and "Postal code" information in the export, the probability of suffering linkage attack will increase [55].

2.3.3 Local and Global Differential Privacy

DP is classified into two types depending on where the noise is added [56]:

1. Local Differential Privacy.

2. Global Differential Privacy.

In local DP adds noise to each individual (input) data point. If we want to find out the average amount in a person's pocket that is sufficient to buy an online course, what are the moderate amounts in people's pockets? But it is possible that many people might not want to give the exact amount! So, what can we do? The local DP system can estimate the amount by asking individuals to add any random value (noise) in the range of -100 to $+100$ to the amount they hold in their pockets and give us just the resultant sum. This means that if 'X' has 30$ in their pocket, we only need to add a random number, say -10, to the total amount $(30 + (-10))$. Hence, we preserve their privacy [57].

As a result, we gain privacy but lose some accuracy as we use averaged values. The higher the privacy protection (plausible deniability), the less accurate the results. This approach's advantage is that the data curator and central aggregator are unaware of the dataset's actual value, and thus confidentiality can be preserved. The user need not rely on the data owner or curator to use the collected data responsibly. In this method, the total noise is much higher since each user must add noise to their data. Often, practical applications must utilize many more users to achieve valuable results. To cope with this issue, practical applications often use high epsilon values (ε) [57].

Globally DP systems are usually more accurate because all the analysis appears on "pure" data. Only a tiny amount of noise needs to be added at the end of the process. However, in global DP, the data owner and every individual has to trust the curator. Local DP is a more conservative, safer model. Supporting local DP, each data point is extremely noisy and not very useful on its own. In a vast dataset, though, the data's noise can be filtered out, and aggregators who collect enough locally private data can properly analyze trends in the whole dataset. In both cases, raw data stay safely protected, and after that, we will do processing on differentially private data. The overview of local and global privacy is shown in Figure 2.5.

So we decided to use global DP for preserving the privacy of the training dataset in our research.

2.4 Machine Learning

Machine Learning is the science of using computers to treat and learn the same as humans and improve their learning skills by updating their information and data by interacting with their environment in an intuitive way [58].

ML systems extract information about the distribution of data. ML systems' main structure gets a set of data called $training - set$ and produces a model based on the features of these data distribution and builds a model to predict unseen data. In the context of DP, we name each sample as $individuals$, and in the concept of ML, we call them $sample$. The whole set of samples in the sample space has the same set of categorical or numerical variables. If the samples in the training set are labeled, and we know, for example, for specific $\mathbf{X}$ based on our trained model, we will produce $\mathbf{Y}$ it would be called supervised learning. In the tasks that we have unlabeled data and want to reach the structure of the data set and decide based on this structure, the model is called unsupervised learning [58].

There are various types of ML algorithms, with hundreds published each day, and either

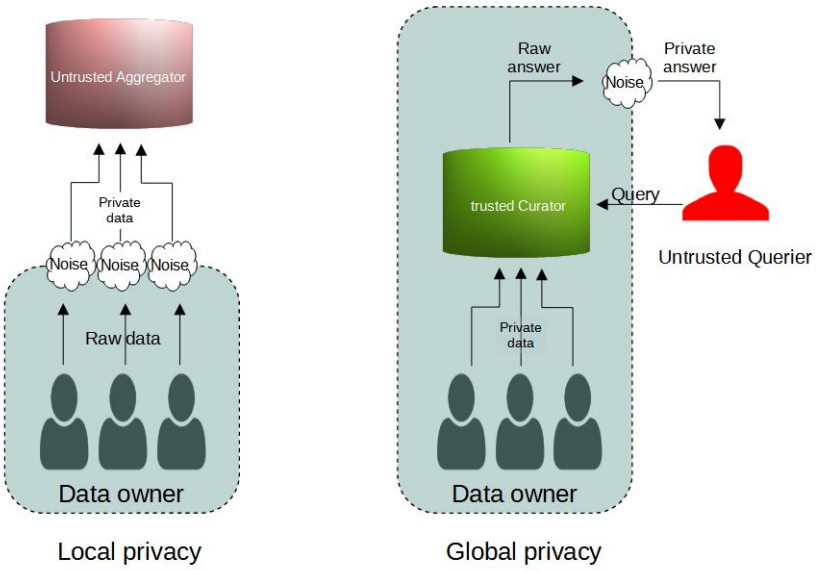


Figure 2.5 The schematic comparison of local privacy and global privacy mechanism. In local privacy (left schematic) the data-owner define the amount of privacy for the data but in global privacy (right schematic), the data owner trust to a curator and the curator define the amount of privacy for data [5].

learning style typically groups them (i.e., supervised learning, unsupervised learning, semi-supervised learning) or by the resemblance in form or function (i.e., decision tree, regression, classification, clustering, deep learning, etc.). Irrespective of learning style or function, all combinations of ML algorithms consist of the following:

1. Representation (a set of classifiers or the language that a computer understands)
2. Evaluation (objective/scoring function)
3. Optimization (search method; often the highest-scoring classifier, for example; there are both off-the-shelf and custom optimization methods)

ML algorithms' critical goal is to generalize beyond the training samples, i.e., successfully interpret unseen data.

2.4.1 How We Get Machines to Learn

There are different methods of getting machines to learn. Using primary decision trees for clustering to layers of artificial neural networks (the latter of which has given way to deep

learning), depending on what task you are trying to fulfill and the type and amount of data you have available. This dynamic sees itself conclude in applications as varying as medical diagnostics or self-driving cars [59] [60].

While the emphasis is often placed on choosing the best learning algorithm, researchers have found that some of the most interesting questions arise from none of the available ML algorithms performing. This problem is with training data most of the time, but this also occurs when working with ML in new domains [60].

Research has done when working on real applications often drives progress in the field, and reasons are twofold [61]:

1. Tendency to discover boundaries and limitations of existing methods
2. Researchers and developers are working with domain experts and leveraging time and expertise to improve system performance.

Sometimes this also occurs by "accident." We might consider the model group or mixture of many learning algorithms to improve accuracy, to be one example. Teams take part in the 2009 "Netflix Price" determined that they got their most reliable outcomes when joining their learning models with other team's learners, resulting in an improved recommendation algorithm [62].

One important point (according to interviews and conversations with experts of ML), in terms of application within a business, health, and elsewhere, is that ML is not just, or even about, automation, an often misunderstood notion. If you think this manner, you are bound to miss the valuable intuition that machines can provide and the resulting occasion (rethinking an entire business model, for example, in industries like manufacturing and agriculture) [63].

Machines with learning ability are beneficial because, with their processing power, they can more quickly highlight or find patterns in big (or other) data that is very hard for humans to detect. ML is a tool that can be used to raise humans' abilities for solving problems and make informed induction on a wide range of problems, from helping diagnose diseases to coming up with solutions for global climate change [63].

2.4.2 Challenges and Limitations

The two biggest, severe problems in ML have involved over-fitting and dimensionality. In over-fitting, the model exhibits bias towards the training data and does not generalize to new data. In dimensionality, when the algorithms with more features work in multiple dimensions,

it will be difficult to understand the data. Having access to a large enough dataset has, in some cases, also been a primary problem [63].

One of the most common mistakes among ML beginners is testing and training data successfully and having the dream of success. Domingo et al. emphasize the importance of keeping some of the data set separate when testing models, only using that reserved data to test a chosen model, followed by learning on the whole data set [63].

When an ML algorithm does not work well, the fastest way to succeed is to serve the ML algorithm more data, which is well-known as a primary driver of the machine and deep learning algorithms in recent years. However, this can lead to scalability issues, in which we have more data but not enough time to learn that data remains an issue [63].

2.4.3 Activation Functions

Deep neural network activation functions are a vital component of deep learning processes. The activation function determines a deep learning model's output, accuracy, and computational efficiency for training. Activation functions affect both the speed of convergence and the likelihood of convergence, or sometimes activation functions might make neural networks incapable of convergent in some cases.

An activation function is a mathematical equation used to determine the output of a neural network. Based on the model's expectations, the function determines whether a neuron should be activated or not based on its input. Besides, activation functions also help normalize every neuron's output to a range between 1 and 0 or between -1 and 1.

Activation functions must also be computationally efficient for each data sample since they are applied across thousands or millions of neurons. Modern neural networks are trained with the back-propagation technique, which places additional computational burdens on the activation function and its derivative.

We have three types of activation functions:

1. Binary step function [64]:

   A binary step function is a function that activates based on a threshold of input. The neuron is activated if the input value falls or rises above a certain threshold and sends the same signal to the network's next layer.

   An issue with a step function is that it does not support classifying the inputs into several categories. It cannot, for example, classify the inputs into more than one category.

2. Linear activation function [64]:

   The output signal is generated by multiplying the inputs by the weight of each neuron. A linear function allows for a greater number of outputs than a step function, which only enables two outcomes, Yes or No. A linear activation function has two major flaws:

   (a) There is no possibility to use back-propagation (gradient descent) to train the model: since the derivative of the function is a constant and has no relationship to the input. Therefore, it is not feasible to understand which weights in the input neurons can provide the best prediction in the future.

   (b) All layers of the neural network merge into one: a layer with a linear activation function is identical to the topmost layer connected to it (because a linear combination of linear functions is still a linear function). Thus a linear activation function turns the neural network into just a single layer.

   A linear activation function is a linear regression model. So the system has limited power and capacity to handle complexity varying parameters of input data.

3. Non-linear activation function [6]:

   Modern neural network models employ non-linear activating functions. They allow the network to generate complex mappings between inputs and outputs, which is essential for modeling and learning complex data, such as images, video, audio, and data sets that are non-linear or have high dimensionality. A neural network can represent almost any process imaginable, provided that the activation function is non-linear. Non-linear functions address the shortcomings of linear activation functions:

   (a) Because they have a derivative function that is related to the inputs, they allow back-propagation.

   (b) They enable creating a deep neural network by stacking multiple layers of neurons. It is necessary to have multiple hidden layers of neurons to learn complex data sets with high accuracy degrees.

Six common activation functions that we use them according to according to the kind of our problem are as below [64]:

1. Sigmoid activation function [64]:

   (a) Pros

i. Smooth gradient.

ii. output values bound between 0 and 1.

iii. Clear predictions:
of X over 2 or below -2 cause the Y value (an estimate) to be at the edge of the curve, at a very close value of 1 or 0. This enables clear predictions of the prediction.

(b) Cons

i. Vanishing gradient:
when the value of X is too high or too low, there is little change to the prediction, leading to a vanishing gradient problem.

ii. Outputs not zero centered.

iii. Computationally expensive.

2. Hyperbolic Tangent activation function [64]:

(a) Pros

i. Zero centered.

ii. Otherwise, it is similar to the sigmoid function.

(b) Cons

i. Like the Sigmoid function.

ii. Computationally expensive.

3. ReLU (rectified liner unit) activation [64]:

(a) Pros

i. Efficient computational design:
Makes it possible for networks to converge as quickly as possible.

ii. Non-linear:
Despite looking like a linear function, ReLU has a derivative function and allows for back-propagation.

(b) Cons

i. The dying ReLU problem:
Occurs when inputs approach zero and are negative, and the network is unable to learn due to the zero gradient.

4. Leaky ReLU(Rectified liner unit) activation function [64]:

   (a) Pros

      i. Prevents dying ReLU problem:
      Because this variant has a small positive slope in the negative region, it enables back-propagation for negative input.

      ii. Otherwise like ReLU.

   (b) Cons

      i. Unfortunately, the results are not consistent enough:
      Fails to make accurate predictions for negative inputs.

5. Softmax activation function [64]:

   (a) Pros

      i. Multiple classes can be handled: The outputs for each class are normalized between 0 and 1, and their sum, which gives the probability of the input value being in a certain class is divided by their sum.

      ii. Useful for output neurons:
      Softmax is commonly only used for neural network outputs, for networks that need to categorize inputs into multiple categories.

2.4.4 Regularization

One of the most critical elements of training your machine learning model is to avoid over-fitting. The model's accuracy will be low if it is over-fitted. This is because your model is trying to capture too much noise in your training data. By noise, we mean the data points that don't represent the data's correct properties but random chance. Learning such information allows your model to become more flexible at the risk of over-fitting. We can use the concept of balancing bias and variance to understand the phenomenon of over-fitting. The effect of regularization can be shown in Figure 2.6.

Regularization is a technique that makes slight modifications to the learning algorithm such that the model generalizes better. This, in turn, improves the model's performance on the unseen data as well as [65].

Definition 14. *regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty* $\Omega(\theta)$*to the objective function J.*

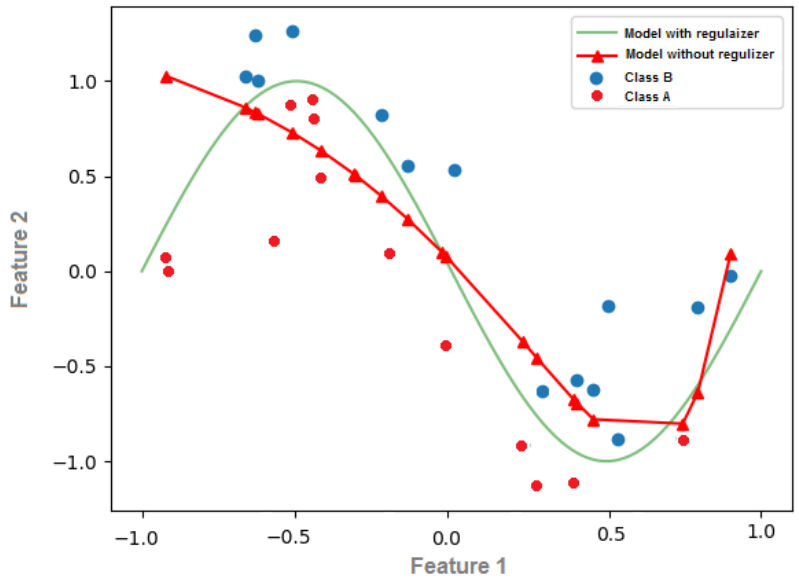


Figure 2.6 The effect of Regularization

We denote the regularized objective function by $\widetilde{J}$:

$$\widetilde{J}(\theta : X, y) = J(\theta : X, y) + \alpha\Omega(\theta) \tag{2.19}$$

Where $\alpha \in [0, \infty)$ is a hyper-parameter that controls the importance (effectiveness) of the regularization term weights, the relative contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization. The vector ω indicates all the weights that should be aected by a norm penalty, while the vector θ denotes all the parameters, including both ω and the regularized parameters [6].

To produce simpler models when you have a large number of features in your data, there are some regularization techniques used to address over-fitting and feature selection [6]:

1. L1 regularization.
2. L2 regularization.

The regularization technique of L1 is referred to as Lasso Regression. The L2 regularization technique is referred to as Ridge Regression.

There is a significant difference in the penalty terms between these two types of offenses.

Ridge regression adds the squared magnitude of a coefficient to the loss function as a penalty term [6].

$$\Omega(\theta) = \frac{1}{2}||\omega||_2^2$$

The objective function with L2 regularization can be re-write as:

$$\widetilde{J}(\theta : X, y) = J(\theta : X, y) + \frac{\alpha}{2}\omega\omega^T \tag{2.20}$$

In L1 regression, we define the penalty term as :

$$\Omega(\theta) = ||\omega||_1$$

The objective function with L1 regularization can be re-write as:

$$\widetilde{J}(\theta : X, y) = J(\theta : X, y) + \alpha||\omega||_1 \tag{2.21}$$

2.4.5 Learning Rate

Putting extra constraints on an ML model to improve the test set's performance (not training set), either by encoding prior knowledge into the model or forcing the model to consider alternative hypotheses that explain the training data [65].

The learning rate schedule changes the learning rate during the learning process. The two important parameters for defining the learning rate are decay and momentum. There are many different kinds of learning rates, but the most popular are time-based, step-based, and exponential [65].

1. Decay serves to settle the learning in a nice place and avoid fluctuation. A situation that may emerge when a too high constant learning rate leads to the learning jump back and forth over a minimum and is controlled by a hyper-parameter.

2. Momentum is analogous to a ball rolling down a hill; we want it to settle at the lowest point of the slope (according to the lowest error). Momentum improves the learning rate (by increasing the learning rate)when the error cost gradient stays the same for a long time and rolls over small bumps to avoid the local minimum. Momentum is controlled by a hyper-parameter analogous to a ball's mass, which must be chosen manually (the start position should be chosen too high). The ball will roll over the minimum, which we wish to find, too low, and it will not fulfill its purpose. The formula for factoring in the momentum is more complicated than for decay but is most often built-in with

deep learning libraries such as PyTorch.

Time-based learning schedules adjust the learning rate based on the learning rate of the previous iteration. Factoring in the decay the mathematical formula for the learning rate is:

$$\eta_{n+1} = \frac{\eta_n}{1 + dn} \tag{2.22}$$

Where η is the learning rate, d is a decay parameter, and n is the iteration step [6].

2.4.6 Batch Size

Batch size Batch size is a term used in ML and refers to the number of training examples utilized in one iteration. The batch size defines the number of samples propagated through the network [65].

1. Batch mode: The batch size is equal to the total dataset, thus making the iteration and epoch values equivalent
2. Mini-batch mode: The batch size is greater than one but less than the total dataset size. Usually, a number can be divided into the total dataset size.
3. Stochastic model: where the batch size is one. The gradient and the neural network parameters are therefore updated after every sample.

Using batch mode has some advantages and disadvantages.

Advantages of using (batch size < number of all samples):

- It requires less memory. Since we train the network using fewer samples, the overall training procedure requires less memory. That is especially important if we cannot fit the whole dataset in our machine's memory [65].
- Typically, networks train faster with mini-batches. That is because we update the weights after each propagation. If we used all samples during propagation, we would make only one update for the network's parameter [65].

Disadvantages of using a (batch size < number of all samples):

- The smaller the batch, the less accurate the estimate of the gradient will be [65].

2.4.7 Optimization

In deep learning, we use optimization algorithms to train neural networks by optimizing the cost function J. By updating the values of trainable parameters of the deep network (ω and b), we minimize the cost function J by using optimization algorithms. We have different kinds of optimizations like SGD, RMSPROB, ADAGRAD, NESTEROV MOMENTUM, MOMENTUM METHOD [65].

We can divide the goal of optimization into two targets.

1. Finding the global minimum of the objective function (The objective function should be convex.)
2. Finding the lowest possible value of objective function according to its neighborhood (It happen when the objective function is non-convex.)

We have three kinds of optimization algorithms:

1. Non-iterative optimization algorithm which is suitable for simple problems.
2. Iterative optimization algorithms regardless of parameter initialization.
3. Iterative optimization algorithms which initialization of parameters is very important for speeding up the convergence speed.

Given an algorithm $f(x)$, with the help of optimization, we can reach to minimum or maximum of $f(x)$. For deep learning, we optimize neural networks by optimizing their cost function J. The cost function is defined as:

$$f(x) = J(W, b) = \frac{1}{m} \sum_{i=1}^{m} L(y^{'i}, y^{i}) \tag{2.23}$$

The value of cost function J is the mean of the loss L between the predicted value $y^{'}$ and the actual value y. The value $y^{'}$ is obtained during the forward propagation step and uses the Weights W and biases b of the network. With the help of optimization algorithms, we minimize Cost Function J by updating the trainable parameters' values W and b [65].

Gradient descent The weight matrix W is initialized randomly. Gradient descent is a first-order iterative optimization algorithm. We use gradient descent to minimize the cost function J and attain the optimal weight matrix W and bias b. Gradient descent is applied

to the cost function J to minimize the cost. The general update rule for gradient descents algorithms can be defined as below. However, in specific they have some difference in updating of parameters, and we will discuss them in detail in the next paragraphs:

$$\text{While to reach maximum/minimum } \begin{Bmatrix} W = W - \alpha \frac{\partial J(W)}{\partial W} \\ \\ b = b - \alpha \frac{\partial J(b)}{\partial b} \end{Bmatrix}$$

The first equation represents the change in weight matrix ω, whereas the second equation represents bias's change of b. The change in the values is determined by learning rate alpha(α) and the derivatives of the cost J concerning the weight matrix ω and bias b. We repeat the update of ω and b until the cost function J has been minimized [65].

Algorithm 2: Gradient descent Algorithm

Result: $\begin{Bmatrix} W = W - \alpha \frac{\partial J(W)}{\partial W} \\ \\ b = b - \alpha \frac{\partial J(b)}{\partial b} \end{Bmatrix}$

Initialization amount of weight and bias

for *i in range of training steps* **do**

forward propagation to calculate X to compute y$'$

;

Calculate cost using Y

Backward propagation to calculate;

$\partial W \partial b$

Updating the parameters W and b

return $\begin{Bmatrix} W = W - \alpha \frac{\partial J(W)}{\partial W} \\ \\ b = b - \alpha \frac{\partial J(b)}{\partial b} \end{Bmatrix}$;

end

Mini-Batch Gradient Descent Gradient descent has the disadvantage that the parameter update is only performed after a full set of training data has been collected. This presents a challenge when the training data is too large to fit in the computer's memory. Mini-batch gradient descent is a very clever workaround for gradient descent that addresses the above issues [65] [51].

Gradient descent with mini-batch distributes the entire training data into small mini-batches of 16, 32, 64, depending on the use case. We then train the network iteratively using these

mini-batches. The use of mini-batch has two advantages:

1. Training starts as soon as we traverse over the first mini-batch, i.e., from the first few training examples.
2. We can train a neural network even when we have a large amount of training data that does not fit the memory.

The batch-size now becomes a new hyper-parameter for our model.

1. When the "batch-size" = "the number of training examples," it is called batch gradient descent (SGD). It faces the problem of beginning learning only after traversing the whole dataset.
2. When the "batch-size" = 1, it is called as Stochastic Gradient Descent. It does not make full use of vectorization, and the training becomes very slow.
3. Therefore, the common choice is 64 or 128 or 256 or 512. However, it depends on the use case and the system memory, i.e., we should ensure that a single mini-batch should fit in the system memory [51].

Algorithm 3: Mini-batch Gradient descent Algorithm

Result: $\begin{cases} W = W - \alpha \frac{\partial J(W)}{\partial W} \\ \\ b = b - \alpha \frac{\partial J(b)}{\partial b} \end{cases}$

Initialization amount of weight and bias

for *i in range of training steps* **do**

  Batches=miniBatchesGenerator(X,Y,batch-size)

  for *j in range of batches* **do**

    minibatchesX,minibatchesY=batches[j]

    forward propagation using minibatchX to calculate y$'$

    Calculate cost using minibatchY

    Backward propagation to calculate ;

    ∂W and ∂b

    Updating the parameters W and b

  end

  return $\begin{cases} W = W - \alpha \frac{\partial J(W)}{\partial W} \\ \\ b = b - \alpha \frac{\partial J(b)}{\partial b} \end{cases}$;

end

Momentum Gradient descent with momentum (it is also called momentum) is an advanced optimization algorithm for optimizing the cost function J. The neural network is trained by updating its trainable parameters using the moving average. [51] [65].

Moving average is the average over n successive values rather than the whole set of values. Mathematically, it is denoted as:

$$V_t = \beta V_{t-1} + (1 - \beta) X_t \tag{2.24}$$

Here, V_t represents the moving average at the t data point for the value X_t. The parameter β controls the value n over which the average is calculated. For example, if $\beta = 0.9$, the moving average considers 10 successive values to calculate the average; if $\beta = 0.99$, the moving average considers 100 consecutive values to calculate the average. Generally, the value n can be approximated by the following formula:

$$n = \frac{1}{1 - \beta} \tag{2.25}$$

It is worthwhile to mention that we can use the mini-batch approach with moment optimizer as only the parameter updating methodology changes with a little difference that we introduce two new variables $V\partial W$ and $V\partial b$ to keep track of the weighted average of the derivatives of

weight ∂W and bias ∂b.

Algorithm 4: Momentum Algorithm

Result: $\begin{Bmatrix} W = W - \alpha V \frac{\partial J(W)}{\partial W} \\ b = b - \alpha V \frac{\partial J(b)}{\partial b} \end{Bmatrix}$

Initialization of weight and bias

for *i in range of training steps* **do**

  Batches=miniBatchesGenerator(X,Y,batch-size)

  for *j in range of batches* **do**

    minibatchesX,minibatchesY=batches[j]

    forward propagation using minibatchX to calculate y$'$

    Calculate cost using minibatchY

    Backward propagation to calculate ;

    ∂W and ∂b

    Updating the parameters W and b

  end

  return $\begin{Bmatrix} V_{\partial W} = \beta V_{\partial W} + (1-\beta)\partial W \\ V_{\partial b} = \beta V_{\partial b} + (1-\beta)\partial b \\ W = W - \alpha V \frac{\partial J(W)}{\partial W} \\ b = b - \alpha V \frac{\partial J(b)}{\partial b} \end{Bmatrix}$

end

RMS-prop RMS-Prop stands for Root-Mean-Square Propagation. The intuition is that division of a large number by another number results in a small number. In our example, the first large number is ∂b, and the second large number that we use is the weighted average of ∂b^2. We introduce two new variables, $S\partial b$ and $S\partial W$, to keep track of the weighted average of ∂b^2 and ∂W^2. The division of ∂b and $S\partial b$ results in a smaller value. The ε is introduced

to avoid the division by 0 error [51] [65].

Algorithm 5: RMS Prob Algorithm

Result: $\begin{Bmatrix} W = W - \alpha.\frac{\partial W}{\sqrt{S_{\partial W}}+\epsilon} \\ b = b - \alpha.\frac{\partial b}{\sqrt{S_{\partial b}}+\epsilon} \end{Bmatrix}$

Initialization of weight and bias

for *i in range of training steps* **do**

  Batches=miniBatchesGenerator(X,Y,batch-size)

  for *j in range of batches* **do**

    minibatchesX,minibatchesY=batches[j]

    forward propagation using minibatchX to calculate y$'$

    Calculate cost using minibatchY

    Backward propagation to calculate ;

    ∂W and ∂b

    Updating the parameters W and b

  end

  return $\begin{Bmatrix} S_{\partial W} = \beta S_{\partial W} + (1-\beta)\partial W \\ S_{\partial b} = \beta S_{\partial b} + (1-\beta)\partial b \\ W = W - \alpha.\frac{\partial W}{\sqrt{S_{\partial W}}+\epsilon} \\ b = b - \alpha.\frac{\partial b}{\sqrt{S_{\partial b}}+\epsilon} \end{Bmatrix}$

end

Adam Adam stands for Adaptive Momentum. It combines both momentum and RMS into one powerful and fast optimizer. We also use error correction to solve the cold start problem in weighted average calculation, i.e., the first few weighted average values are too far from their real values. The V value is derived from momentum, while the S value is derived from the RMS formula. It is important to note that we employ two different values. of β during the calculations. $\beta 1$ is used for momentum calculations, whereas $\beta 2$ is used for RMS-prop calculations. We can still use the mini-batch approach with Adam optimization as only the

parameter update method changes [51] [65].

Algorithm 6: ADAM Algorithm

Result: $\begin{cases} W = W - \alpha.\frac{V_{\partial W}^{corrected}}{\sqrt{S_{\partial W}^{corrected}}+\epsilon} \\ b = b - \alpha.\frac{V_{\partial b}^{corrected}}{\sqrt{S_{\partial b}^{corrected}}+\epsilon} \end{cases}$

Initialization of weight and bias

for *i in range of training steps* **do**

Batches=miniBatchesGenerator(X,Y,batch-size)

for *j in range of batches* **do**

minibatchesX,minibatchesY=batches[j]

forward propagation using minibatchX to calculate y$'$

Calculate cost using minibatchY

Backward propagation to calculate ;

∂W and ∂b

Updating the parameters W and b

end

return $\begin{cases} V_{\partial W} = \beta_1 V_{\partial W} + (1-\beta_1)\partial W \\ V_{\partial b} = \beta_1 V_{\partial b} + (1-\beta_1)\partial b \\ V_{\partial W}^{corrected} = \frac{V_{\partial W}}{1-\beta_1^i} \\ V_{\partial b}^{corrected} = \frac{V_{\partial b}}{1-\beta_1^i} \\ S_{\partial W} = \beta_2 S_{\partial W} + (1-\beta_2)\partial W^2 \\ S_{\partial b} = \beta_2 S_{\partial b} + (1-\beta_2)\partial b^2 \\ S_{\partial W}^{corrected} = \frac{S_{\partial W}}{1-\beta_2^i} \\ S_{\partial W}^{corrected} = \frac{S_{\partial b}}{1-\beta_2^i} \\ W = W - \alpha.\frac{V_{\partial W}^{corrected}}{\sqrt{S_{\partial W}^{corrected}}+\epsilon} \\ b = b - \alpha.\frac{V_{\partial b}^{corrected}}{\sqrt{S_{\partial b}^{corrected}}+\epsilon} \end{cases}$

end

2.4.8 Gradient Clipping

Neural networks work very well for solving the problems, but in some cases, it may cause a problem in training: exploding gradient.

Gradient clipping is a method for overcoming this defect of neural networks. The main idea about this method is that when the gradient gets too big, we re-scale it and keep it small. For this reason, we set a threshold as C and if $||g|| \geq C$, then:

$$g \leftarrow C.\frac{g}{||g||}$$

where g is gradient, C is a threshold and this is one of our hyper-parameter, and $||g||$ is the norm of g. If $||g|| \leq C$ we do not need to do anything.

This method will guarantee that the gradient has the norm at most C. This way, it ensures that the gradient descent will have proper behavior even if the model's loss landscape is irregular. The following Figure shows the action of clipping on loss function with a cliff and shows that parameters stay in the right region with clipping because the descent step size is limited and non-reasonable behavior without clipping because they may get colossal descent step toward the optimal solution and it farther them from the optimal solution [66].

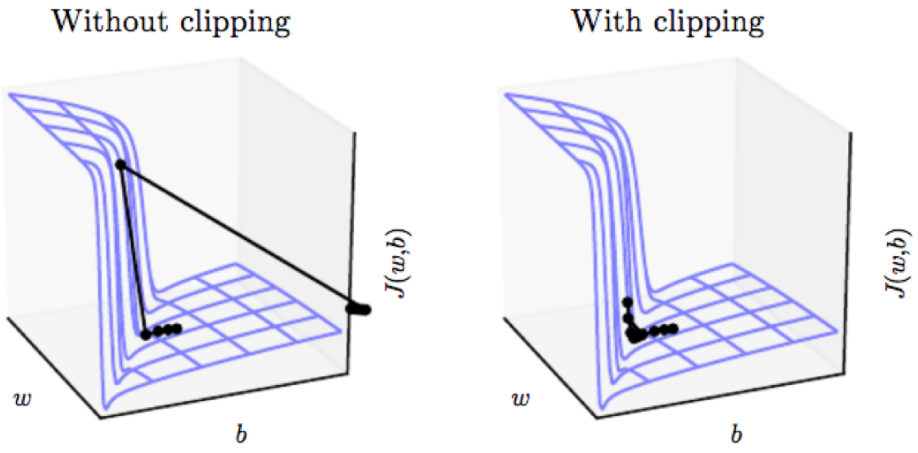


Figure 2.7 The right image is the effect of gradient clipping which it has a limited descent step size and the left image is the same problem without clipping which it has a non limited descent step size [6].

2.5 Deep Learning

A deep neural network is one of ML's subsets, which is remarkably useful for ML tasks that define a function based on input and produced output by combining many layers of basic building blocks.

Artificial intelligence is basically when machines can perform tasks that usually require human intelligence. It encloses ML, where machines can learn by experience and accumulated skills without human participation. Deep learning is a subset of ML where artificial neural networks, algorithms inspired by the human brain, learn from big data. The deep learning algorithm, similar to how we learn from experience, repeats a task repeatedly, each time tweaking it to improve the outcome. We refer to 'deep learning' because the neural networks have various (deep) layers that enable learning. Just about any problem requiring "thought" to Figure out is a problem deep learning can learn to solve [67].

Even when using diverse, unstructured, and interconnected data, deep learning can assist machines in solving complex problems. As much as the deep learning algorithms learn, they can perform better [67] [68].

2.5.1 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a category of Neural Networks with outstanding performance in complex tasks such as image recognition and classification. ConvNets have been very successful in face recognition and identification, and machine vision is significant in robots and self-driving cars.

The first architecture of CNN was called LeNet, mainly used for character recognition tasks like zip codes, digits, etc. In Figure 2.8 we can see a sample of CNN that classifies an input image into specific categories like a cat, dog, etc., and the highest probability is 94% among all four categories [69].

As we can see in Figure 2.8, we have four main operations in CNN, which are the basic building blocks of every Convolutional Neural Network as below [69].

1. Convolution.
2. Activation function.
3. Pooling (Sub-sampling.)
4. classification.

In CNN, we work with images. Every image can be presented by a matrix that each matrix element is a pixel value as we see a sample in Figure 2.9.

Channel is the conventional term used to refer to a particular component of an image. A standard digital camera's output image will have three channels of red, green, and blue. We can imagine those three Chanel as three 2d-matrices stacked over each other, and each is having pixels with values in the range 0 to 255.

A grayscale image is not like an output image of the camera, consisting of 3 layers and only one channel. We will only consider grayscale images from the MNIST dataset for our work so that we will use only one 2D matrix to represent an image. Each pixel's value in the matrix will range from 0 to 255, that zero indicating black and 255 indicating white.

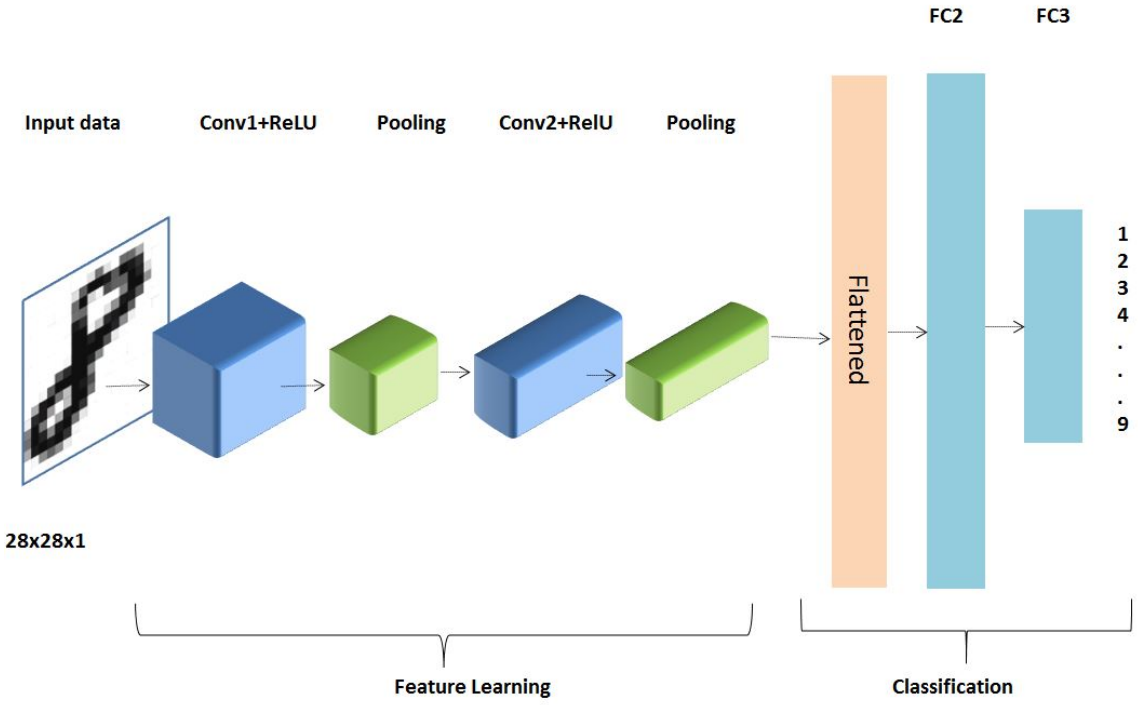


Figure 2.8 A simple CNN for classification of handwritten digits with two convolutional layers and two fully connected layers. We have categorized it into two parts of feature learning and classification part.

Convolution Convolution's primary purpose in the case of a CNN is feature extraction from the input image. The convolution process preserves the spatial relationships between pixels by learning the image features using small input data squares. We will cover the entire image by convolution operation and produce a convolved feature, or feature map [69, 70].

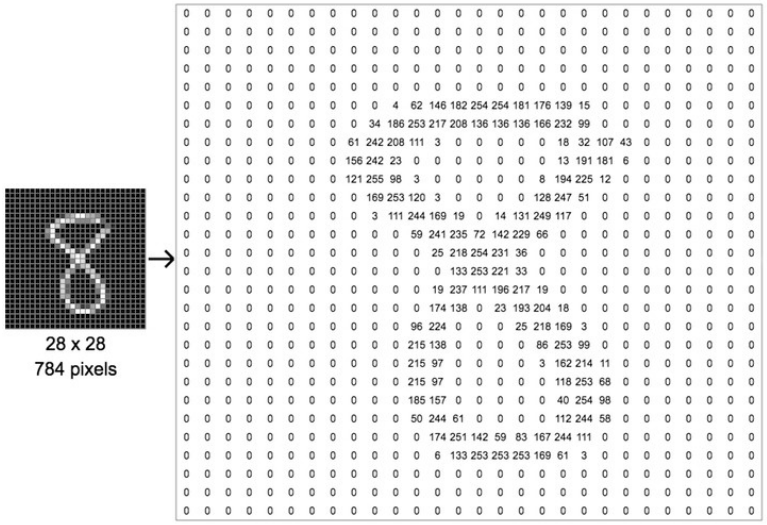


Figure 2.9 Image of a handwritten eight from MNIST dataset, each pixel value of this image is a number from the range of [0,255] according to each pixel's intensity. 0 is an entirely black pixel, and 255 is the white one. The other values between 0 and 255 are gray pixels. The right side image is the associated intensity matrix of the picture.

Pooling Spatial pooling (sub-sampling or down-sampling) reduces each feature map's dimensionality but retains the most crucial information. There are different types of spatial pooling like Max pooling, Average pooling, Sum pooling, etc.

In max pooling, the maximum element of the rectified feature map from a local neighborhood (for example, a 2×2 window) is considered. Instead, we could take the average (Average Pooling) or sum of all window elements rather than the largest element. In practice, it has been demonstrated that max-pooling works better.

The function of pooling is to reduce the spatial size of the input representation progressively. It will make the input representations (feature dimension) smaller and more manageable, makes the network invariant to little distortion, transformation, and translations in the input image, and helps us arrive at an almost scale consistent representation of our image.

Fully Connected Layer The Fully Connected layer is a multi-layer perceptron that uses a softmax activation function in the output layer. Using the term "Fully Connected" is because every neuron in the next layer is connected to every neuron on the previous layer [69, 70].

Convolutional and pooling layer output represents the high-level features of the input image. With the help of the fully connected layer, we use the previous layer's high-level features for classifying the input image into various classes according to the training dataset.

Beyond the classification, we can learn non-linear combinations of features at a low cost by adding a fully-connected layer. In convolutional and pooling layers, most of the features may serve the classification task well. However, a combination of those features may serve the classification task even better. Here is an overview of the convolution network training process as follows [69, 70]:

1. Initialization of all filters and parameters/weights with random values
2. Taking a training image as input goes through the forward propagation step (first, in convolution layer, second, ReLU, and third, pooling operations along with forwarding propagation in the fully connected layer) and finally, finds the output probabilities for each class.
   (a) Lets say the output probabilities for the boat image above are $[0.2, 0.4, 0.1, 0.3]$.
   (b) Since weights are randomly assigned for the first training example, and output probabilities are also random.
3. Calculation of the total error at the output layer.
4. Use back-propagation to calculate the gradients of the error concerning all weights in the network and use gradient descent to update all filter values/weights and parameter values to minimize the output error.
5. Continuously repeats steps 2,3 and 4 with all the training set images.

2.5.2 VGG-net

K. Simonyan and A. Zisserman from the University of Oxford proposed VGG-net as a model for convolutional neural networks in a paper titled "Very Deep Convolutional Networks for Large-Scale Image Recognition." VGG was a neural network that accomplishes very well in the Image Net Large Scale Visual Recognition Challenge (ILSVRC) in 2014. It scored first on the image localization task and second place on the image classification task [71].

Localization is the process of finding where a specific object is in the image based on the bounding box. Classification is expressing the item in the picture. This predicts a category label, such as "cat" or "bookcase."

The model achieves top-five accuracy ratings of 92.7% in ImageNet, a dataset consisting of over 14 million images belonging to 1000 classes. This was one of the well-known models presented at the ILSVRC-2014. VGG-net improves the Alex-net model by replacing large

kernel-sized filters (11 for the first layer and 5 in the second convolutional layer) with multiple 3×3 kernel-sized filters one after another. Figure 2.10 shows the architecture of this deep neural network, and in the table 2.3, we can see each layer's detail. We use C for the convolutional layer and P for the pooling layer [71].

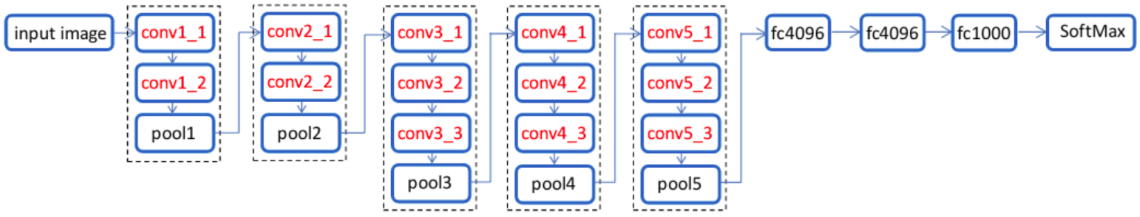


Figure 2.10 A block-based architecture of VGG with 13 convolutional layers, five max-pooling layers, and three fully-connected layers net.

Layer size and stride									
Layer	$C1_1$	c1-2	P1	C2-1	C2-2	P2	C3-1	C3-2	C3-3
Size	3	5	6	10	14	16	24	32	40
Stride	1	1	2	2	2	4	4	4	4
Layer	p3	c4-1	c4-2	C4-3	p4	c5-1	C5-2	C5-3	p5
Size	44	60	76	92	100	132	164	196	212
Stride	8	8	8	8	16	16	16	16	32

Table 2.3 The complete detail and size of 13 convolutional, five max-pooling, and three fully-connected layers of Vgg-net.

2.5.3 Alex-net

Convolutional neural networks (CNNs) were traditionally used in object recognition. They are powerful models that are very easy to train and control. Since they are employed on millions of images, there is no over-fitting at an alarming scale. The performance of these neural networks is relatively the same as that of standard feed-forward neural networks. The only problem is that they are hard to apply to high-resolution images. At the ImageNet scale, a GPU-optimized training program would reduce the training time while enhancing the performance. The architecture is composed of eight layers: five convolutional layers and three fully-connected layers. However, this is not what makes Alex-net special; Alex-net uses a few advanced features, including convolutional neural networks (CNNs) as below [72]:

1. ReLU Non-linearity. AlexNet uses the Rectified Linear Unit (ReLU) in place of the standard *tanh* function. *ReLU* offers a significant advantage in terms of training time. A CNN trained with *ReLU* reached a 25% error rate on the CIFAR-10 dataset six times faster than a CNN trained with *tanh*.

2. Multiple GPUs. Back in the day, GPUs had 3 gigabytes of memory (today, such numbers would be considered rookie numbers). AlexNet allows training via multi-GPU by distributing the model's neurons between two GPUs. Not only does this reduce training time, but it also allows a larger model to be trained.

3. Overlapping Pooling. CNNs typically pool the output of neighboring groups of neurons without overlap. However, when the authors introduced overlapping, they observed a reduction in error of about 0.5% and found that overlapping pooling models are generally more difficult to over-fit.

Alex-net is a powerful model capable of achieving high accuracy on very challenging datasets. However, removing any of the convolutional layers significantly decreases Alex-net's accuracy. Alex-net is a leading technology for object-detection tasks and has tremendous potential in the field of artificial intelligence problems using computer vision [72]. The architecture of Alex-net is shown in figure 2.11.

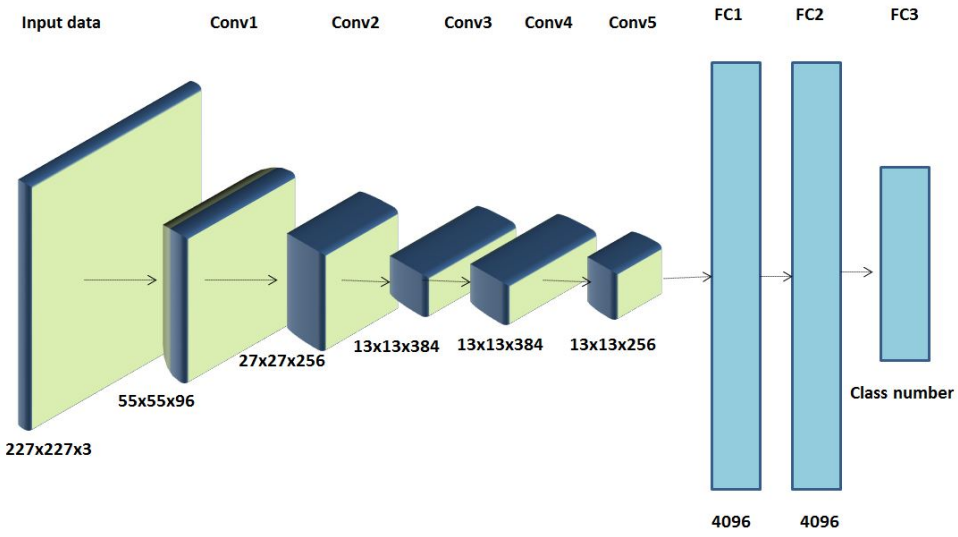


Figure 2.11 Illustration of Alex-net's architecture. Image credits to Krizhevsky et al., the original authors of the Alex-net paper.

2.6 Statistical features of data

This section reviews the statistical features for data perturbation efficiency while we add noise to it. When we add noise to our data, the perturbed data should keep the same statistical properties as the original data because we want to learn and extract features from it. According to this fact, consideration has to be made for statistical characteristics such as normal distribution, mean, variance, covariance, standard deviation, and correlation of original dataset and differential private dataset.

2.6.1 Mean

The Mean μ is the average of values after their total sum has been taken. For this purpose, we would look at the summation of values. Then we divide them by the n, the number of values; the mathematical statement then for the mean μ, is below [73]:

$$\mu = \frac{1}{n} \sum_{k=0}^{n} x_i \tag{2.26}$$

2.6.2 Variance

In statistics science, the variance is the expectation of a random variable's squared deviation from its mean. In other words, it measures how far a set of numbers is spread out from their average value. The variance, which we show by the parameter of σ^2, in noise addition, is a measure to know how data distributes itself in approximation to the mean value. The formula for calculating the variance is as below [74]:

$$\sigma^2 = \frac{\sum((X - \mu)^2)}{N} \tag{2.27}$$

2.6.3 The Standard Deviation

The Standard Deviation, σ is a measure of how distributed data is from the normal. Afterward, we would say standard deviation is how data points deviate from the mean. The mathematical expression is simply the square root of the variance σ^2, and we can calculate it as below [75]:

$$\sigma = \sqrt{\sigma^2} \tag{2.28}$$

2.6.4 Covariance

When we added noise based on Gaussian or Laplacian distribution, the measurement of original data affiliation and perturbed data is crucial. The covariance, $Cov(X, Y)$ metric, calculates how affiliated the deviations between the data points X and Y are. When the covariance is positive ($Cov(X, Y) > 0$), then the X and Y data points tend to increase together. When the covariance is negative ($Cov(X, Y) < 0$), then the tendency is that for the two data points X and Y, the greater values of one variable mainly correspond to the lesser values of the other, when the covariance is zero, this will signal that the data points are independent. The expression for covariance is given as below [76]:

$$cov(x, y) = \frac{1}{N} \sum x_i y_i - \bar{x}\bar{y} \tag{2.29}$$

2.6.5 Correlation

The correlation r_{xy} calculates the capability and tendency of an additive or linear relation between two data points X and Y. The correlation r_{xy} is dimensionless, independent of the parts in which the data points x, and y are calculated. For analyzing the result of correlation, we have three result mood:

1. If $r_{xy} = -1$, then r_{xy} shows a negative linear relation between the data points x and y.
2. If $r_{xy} = 0$, then the linear relation between the two data points x, and y does not exist. However, a regular nonlinear relation might exist.
3. If $r_{xy} = +1$, then there is a strong linear relation between x and y .

The expression for covariance is given as below [76]:

$$correlation = r_{xy} = \frac{cov(x, y)}{\sigma_x \sigma_y} \tag{2.30}$$

2.6.6 Normal Distribution

Where σ^2 is variance, μ is the mean, X is the single data values, and N is the number of values.

The Normal Distribution, also known as the Gaussian distribution, used in calculating the noise addition, is a bell-shaped continuous probability distribution used to demonstrate real-valued stochastic variables that accumulate around a single mean. The annotation $N(\mu, \sigma^2)$

represents a normal distribution with mean μ and variance σ^2 . The formula for normal distribution is below [77]:

$$f(x) = \frac{1}{\sqrt{(2\pi\sigma^2)}} e^{-(\frac{(x-\mu)^2}{2\sigma^2})} \tag{2.31}$$

The parameter μ represents the mean, which is located in the center of the bell shape diagram, the parameter σ^2 represents the variance, which is the width of the distribution.

2.6.7 Signal Noise Ratio (SNR)

In signal processing, the concept of the signal-noise ratio is used to calculate a signal contaminated by noise according to approximating the signal power to noise power ratio, fundamentally we calculate it by the ratio of the power of the signal without noise over the power of the noise.

In our work, we use a signal-noise ratio to achieve optimal data efficiency. At the same time, we preserve the privacy of information by measuring the amount of noise we can add to our data to have optimal perturbation and obfuscation. In signal processing, the concept of the signal-noise ratio is used to calculate a signal contaminated by noise according to approximating the signal power to noise power ratio, fundamentally we calculate it by the ratio of the power of the signal without noise over the power of the noise [78].

The formula for calculating the signal-noise ratio is below:

$$Signal\ noise\ ratio = \frac{Signal\ variance}{Noise\ variance} \tag{2.32}$$

In the image processing field, we employee the signal-noise ratio and use the ratio of the mean pixel value to the standard deviation of the pixel values in a particular vicinity as below [78]:

$$Signal\ noise\ ratio = \frac{mean\ pixel\ value}{standard\ deviation\ of\ the\ pixel\ values} \tag{2.33}$$

CHAPTER 3 PROPOSED METHOD FOR DIFFERENTIAL PRIVATE DEEP NEURAL NETWORK

In this chapter, we discuss the research methodology of our study. Our proposed method is constructed of three main phases as shown in Figure 1.2. **Phase I (Pre-Processing)**, of the approach, is responsible for performing a reprocessing to preserve the privacy of the training set. In **Phase II (Database partitioning and DPAdam)**, we split the pre-processed dataset and insert them as input to our differential private learning algorithm, as we call it DPAdam. This splitting aims to increase our system's privacy, and we use the ensemble technique to make aggregation between these different learning models.

With the intent to increase our system's total privacy and prevent API attacks by making the first and second phases as a black-box system **Phase III (API hardening)** is focused on API hardening inputs the predicted results of **Phase II** and trains a new model. The output of **Phase II** combined with the test set (input of **Phase II**) forms the training set for the new model of **Phase III**.

Further, we evaluated our proposed method in two ways:

First, for illustrating the effectiveness of using protected data for training a DNN, we used synthesized (and randomly generated) data. This data has a simple structure of two classes being denoted as A and B. We measured the effectiveness of our method by analyzing the relation between protection and the dataset's size.

Second, for testing the performance of our DNN with privacy protection mechanism, we used the MNIST (state of the art dataset of handwritten digits) [79].

All the parts are explained in detail in this chapter. The process diagram of our proposed method is visualized as we see in Figure 3.1.

3.1 Phase I. Pre-processing

In ML, data pre-processing is a vital step that helps increase data quality to boost meaningful insights from the data. Data pre-processing in ML refers to preparing (cleaning and organizing) the raw data to make it suitable for ML models' training. In this thesis, we extended the definition of pre-processing by additional security concerns. Hence our definition slightly differs from the traditional definition. By combining DP and the anonymization technique,

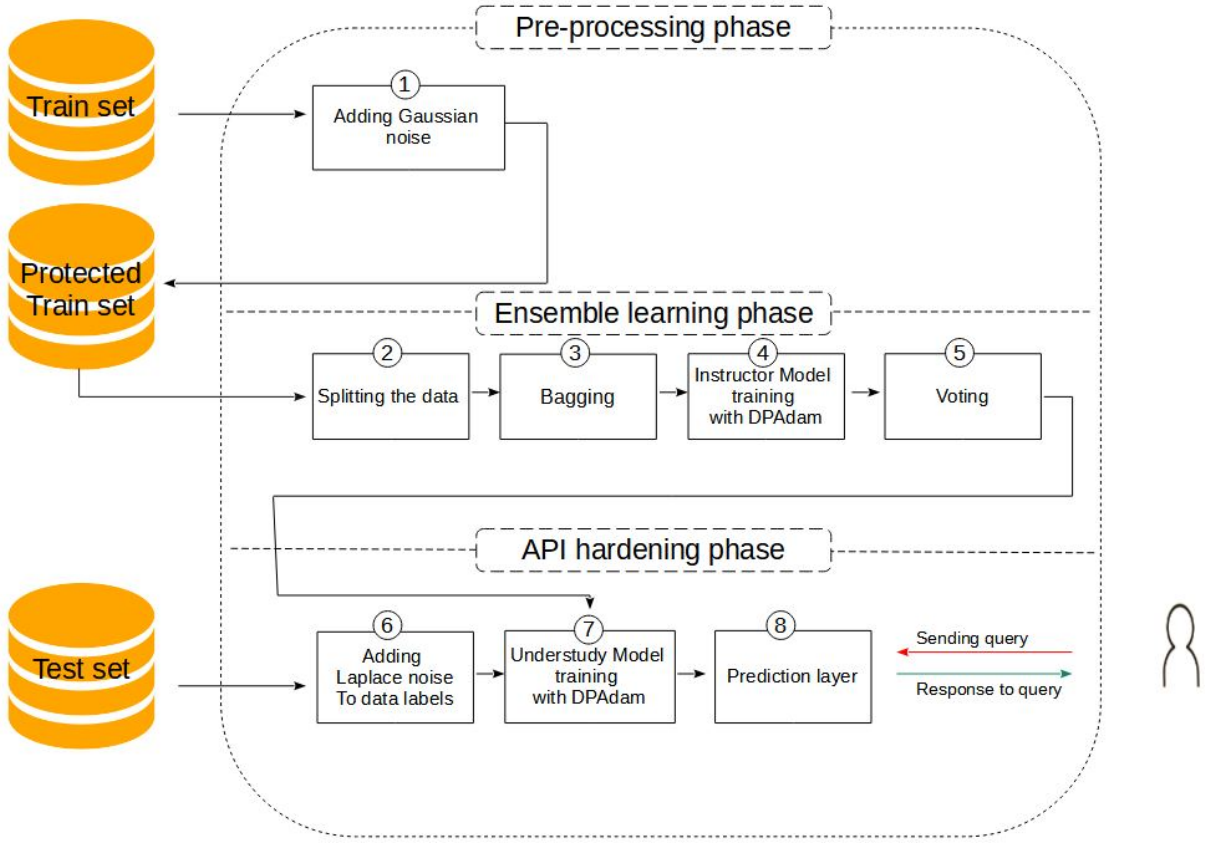


Figure 3.1 The process diagram of project, the protected training dataset is the MNIST training-set, and the test set is the test-set of MNIST dataset. We use $\frac{9}{10}$ of test-set to train the understudy phase and $\frac{1}{10}$ to test this phase.

we pre-process our primary data to preserve information privacy. Our pre-processing phase has one-step steps **(i) protecting the privacy with Gaussian mechanism** to guarantee that our training dataset is (ε, δ)-DP.

While DNN provide a powerful tool to achieve multiple tasks, they have difficulty handling noise which is common in real-world testing. When training neural networks, we strive for maximum accuracy during training. But in most cases, what matters most is the generalizability of the neural network model. The accuracy of the neural network results is hugely dependent on the quality of the training set and the similarity and homogeneity of the data to the training set.

Training a neural network involves several pre-processing activities, including but not limited to resizing, mean normalization, and subtracting RGB value before inputting it into the model. However, in most cases, one does not consider the effects of noise on data. This

causes failure in the model's generalizability [80].

One approach to overcome the generalizability problem and improve the robustness of neural networks is to train the model with random noise applied to their inputs [6]. This noise is in the form of random data addition to the training data inputting the model.

The study of Nazare et al. [81] indicated that training networks using data with varying noise levels could be beneficial for applications that must deal with images of different quality because it improves the network's resilience to other types of noise and noise levels. These benefits include:

1. Increasing the model's generalizability,
2. Lower risk of over-fitting during training,
3. Can be used as a data augmentation technique in the cases in which the number of our training data set is small.

The lower risk of over-fitting is also consistent consistently leads to better generalization. Less over-fitting results in better validation and test scenarios reflected in better generalization during real-world testing.

DP works by adding statistical noise to data (either to its inputs or to its output). DP ensures that a third party cannot reliably infer whether a particular individual is actively participating in a database query. This does not pose any limitation on the access to unlimited computational power and access to every entry in the database except for that individual's data [12]. We discussed the difference between global and local differential privacy in Chapter 2. We use global differential privacy as we assumed that the global trust to the curator of the noise is possible in our system for simplicity reasons in trade for high model accuracy.

3.1.1 Step 1. Gaussian Differential Privacy

In addition to anonymization to preserve the data privacy, we integrated the use of Gaussian DP [28] into the process. We use Gaussian DP to add perturbation in a range of (0, 10]. The lower amount of ε (closer to zero), the more noise would be added to the data that in turn increases the privacy we are targeting. However, Gaussian noise addition considers the outliers and the most influential observation. The higher amounts of noise can make the data useless. The relation between ε and the amount of noise we add to our data is reverse. It means that less amount of ε means adding more noise to the data, which in turn creates more privacy at this level(as much as we increase the amount of noise, we will have more privacy), but the more amount of noise (smaller ε) only renders the data useless.

The main benefit of using Gaussian noise (and hence creating Gaussian-DP) is that when our data are sensitive by using this method of DP, we do not perturb the information a lot to lose their statistical meaning. On the other hand, we have also preserved the privacy of our data. When we increase the number of entries in the dataset, it makes sense that by preserving privacy, we have good accuracy in extracting statistical information from our dataset [82,83]. Thus, the balance between privacy and efficiency is still a challenge.

One significant advantage of Gaussian noise is that the distribution itself behaves well. The Gaussian distribution is more concentrated and its' converge to zero faster and it will make the variance of additive noise will be less than Laplace mechanism. The normal distribution is called so because it is convenient, and it is used in both the natural and social sciences widely. The method is frequently used to model unknown random distributions. When you add up many independent random variables, you will arrive at a normal distribution. These are only some of the properties of this fundamental distribution. Many experts assume an understanding of Gaussian noise in data analysis and scientific research. Since anonymized statistics are released, analysts don't need to learn too many new concepts to understand how these data are protected [12, 84].

The other advantage of using a Gaussian distribution is that its tails are very narrow. A significant proportion of the probability mass of the distribution is concentrated around its mean value.

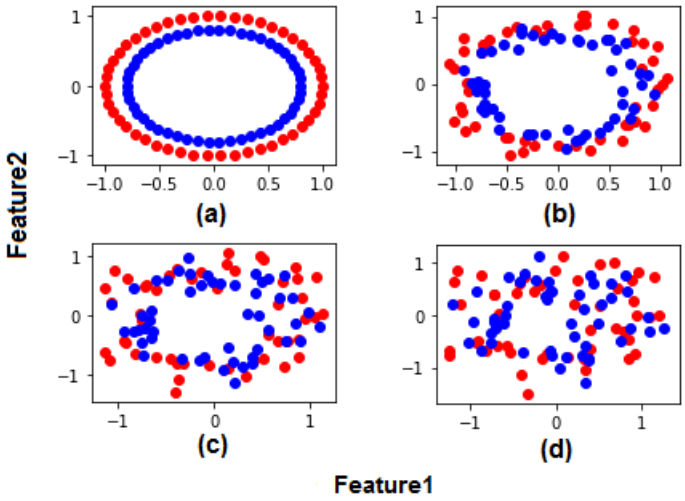
Consider a normal distribution with mean 0 and standard deviation σ. Based on the 68–95–99.7 rule, random variables sampled from this distribution will look like [85]:

1. in [σ,σ] with 68% probability.
2. in [2σ,2σ] with 95% probability.
3. in [3σ,3σ] with 99.7% probability.

Furthermore, another excellent feature of Gaussian noise when a single input of a dataset impacts k distinct statistics is necessary to scale the Laplace noise factor by k. In comparison, Gaussian noise can be scaled by only $\sqrt{k}$. Comparing them yields a more positive view of Gaussian noise's power [12, 84].

When we have one statistical feature, it is better to use Laplace mechanism and it is also more powerful than Gaussian mechanism. If it can impact many, and we have more than one statistical feature, like images, Gaussian mechanism is better. As we want to use Gaussian noise to preserve privacy in the machine learning task, each data point influences the vector's coordinates at each iteration. For having a differentially private model, all the coordinates

must be subject to noise. So the good choice would be Gaussian noise for precisely the same reasons [12, 84].



(1).png (1).png (1).png (1).png

Figure 3.2 Different amount of added noise, in (a) we have pure dataset with no privacy, (b) we have perturbed the dataset with Gaussian noise N(0,0.2), (c) we have perturbed the dataset with Gaussian noise N(0,0.4), (d) we have perturbed the dataset with Gaussian noise N(0,0.6).

For illustrative purposes, We used a straightforward classification problem with two classes. In Python, using the **Scikit-learn** library provides the **make_circles()** function, which can create binary classification problems with prescribed numbers of samples and statistical noise.

Examples provide input variables that represent the x and y coordinates of the point on a two-dimensional plane. The points are organized into two concentric circles (their centers are the same) for the two classes. A Gaussian noise can be included when sampling the points with the "noise" argument that defines the noise's standard deviation. The seed for the **pseudorandom number generator** [86] is specified via the *random_state* argument, which is used to sample the same points each time the function is called.

Next, at first, 100 data points are generated, and the number of data points is increased to examine the impact of the number of inputs on the classification when the number of inputs is increased. A seed of one is used to seed the pseudorandom number generator. The input variables have x and y values centered at 0.0 and the bounds $[-1, 1]$. Besides, the class values are integer values for either A or B, and that examples are shuffled between the classes.

We used a "toy" machine learning problem and to demonstrate the process of adding Gaussian

noise. We used a randomly generated data set in two different classes naming A and B 25 nodes in the hidden layer and one output. We used a Rectified Linear Unit (ReLU) activation function [87] for the nodes in the hidden layer. We also used the Sigmoid activation function [88] is used for binary classification to predict the probability of a sample belonging to two classes A and B. We used each class of A, and B is in the range of [-1 1]. As a result, we have a binary classification problem. We randomly generated data points. We call this data set DILL[1]. Figure 3.2- (a) demonstrates the distribution of these data points on two dimensions (A and B). This data set is the input of our model, and we are aimed to improve the privacy of it. Figure 3.2- (b),(c), and (d) demonstrate three different amounts of noise added to the below formulation. The result of these protected dataset training is in the 4th chapter.

Following described method in this subsection, we add Gaussian noise into DILL to have a (ε, δ)-differential private data set according to below equation.

$$\varepsilon = \sqrt{2log(\frac{1.25}{\delta})} \times \frac{\Delta_2 f}{\sigma} \tag{3.1}$$

We used Gaussian function (Gaussian distribution) to generate a randomized number with $mean = 0$ and the $\sigma = 0.6$,we will have ($\varepsilon = 2.5$,$\delta = 10^{-5}$)-DP dataset. Then we add this noise to each data input if DILL:

$$primary\ data + noise\ from\ Gaussian(0, 0.4) = perturbed\ result$$

Gaussian noise also acts as a regularizer [89, 90] and prevents the model from over-fitting a pre-processing step, and we can have a robust model [91]. The use of Gaussian-DP makes the model more generalizable and robust [6, 91]. Using Gaussian-DP keeps the statistical features of the dataset when being used on a large-scale data set as shown in Figure 3.3.

With the help of regularization, we add a penalty to our cost function to prevent models' over-fitting in ML. We should ensure our model will perform well on unseen data during our system's learning and training. Hence we add a penalty as an extra constraint to our model for increasing the performance. Therefore, we use a regularizer to get closer to our optimal target because it will increase the reduced variance bias [6, 91].

By this step of privacy preservation, we also have prevented our training model from the over-fitting problem.

[1]DILL stands for Data ILLustration

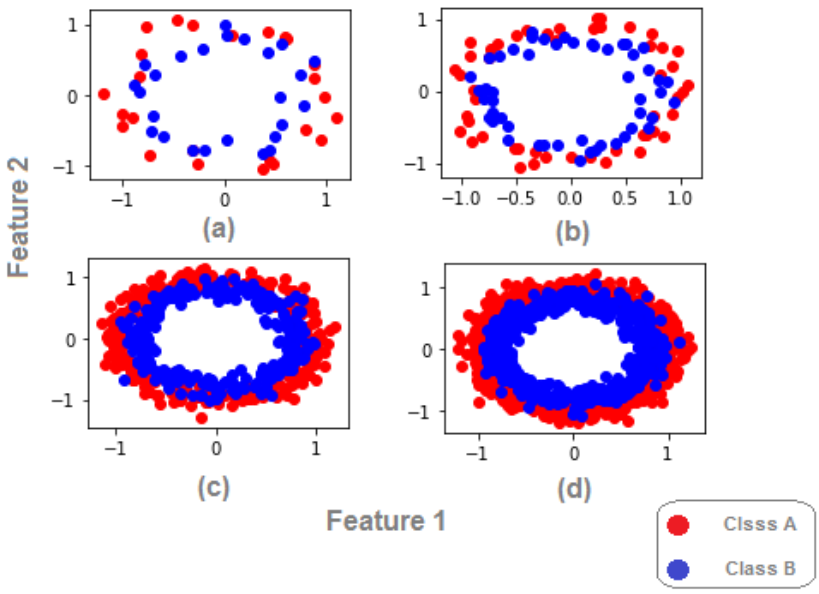


Figure 3.3 The effect of added noise on different number of dataset entries, in (a) we have dataset with 50 entries, (b) we have dataset with 100 entries, (c) we have dataset with 500 entries, (d) we have dataset with 1000 entries.

3.2 Phase II. Ensemble Learning

In the next phase, we start by partitioning the private dataset into subsets of data. There is no overlap between each data partition, and we split our training set into N different categories. If a record was in one of the primary data subsets, it is included exactly in that partition and does not exist in other sub-sets. We train an ML model, called an Instructor, on each of these partitions. There is no limitation on how the instructors are trained, and this is one of the main advantages of our proposed architecture. All of the Instructors solve the same ML problem, but they were trained independently apart from other learning models or algorithms. This technique is called the ensemble technique. The reason for partitioning is to increase the privacy of information. If an attacker penetrates our system, they will extract just one of the instructors' parameters or training set. As much as we increase the number of instructors, privacy will increase. But on the other side, when we increase the number of partitioning in the training set, each of these instructors will not perform well because of the fewer number of training sets than with one partition. So there should be a trade-off between the number of instructors. In our research, we have chosen ten instructors and analyzed the performance of the model. Because our training dataset (MNIST) consists of 60000 images [30] for the training set, suppose we split the dataset into many partitions.

In that case, each DP-training algorithm's number of the training set will decrease because each division is independent of others and does not overlap with other partitions. Therefore, our model will not train well, although we have increased the privacy.

3.2.1 Ensemble Learning

In ensemble learning, multiple models are combined to produce a robust model that is often much more accurate and powerful than anyone model used individually [92]. We can see the process diagram for this part in Figure 3.1 in the ensemble learning phase.

The choice of the model in ML is crucial to obtaining high accuracy predictions. Many factors affect this decision, including the quality of data, the dimensionality of space, etc. Our data's quality and purity are certainly degraded when we add noise to protect our confidential information in our pre-processing phase. We need to increase the accuracy of the model prediction by using the ensemble method. Bias and variance are the two features of a model that make up its fundamental properties, although they usually vary in opposite directions. For training a model, we need to consider the trade-off between bias and variance. To "solve" a problem, we need our model to have enough degrees of freedom to resolve the underlying complexity of the data we are working on them. However, we must avoid overly large degrees of freedom at the same time to be more robust [92].

A basic model in the ensemble learning context can serve as a building block for designing more complex models by combining several basic models. Based on the available data, it can be seen that these basic models do not perform well because they have either too much variance or are biased. By combining several of these basic models into one strong learner (ensemble model), the aim is to reduce bias and/or variance to achieve better performances [92, 93].

We need to select our base models to be aggregated to set up an ensemble learning method. It is common to use a single base learning algorithm when training weak learners. This results in a more homogeneous set of weak learners than when using multiple base learning algorithms. As a consequence, the ensemble model we obtain is said to be homogeneous. However, some methods use various learning algorithms, such as heterogeneous weak learners combined to make a "heterogeneous ensembles model." One crucial point is that our weak learners' choices should be consistent with how we aggregate these models. If we pick models with low bias and high variance, we should choose an aggregation method to reduce variance. On the other hand, if we prefer models with low variance and high bias, we should select an aggregation method that reduces bias.

Three main types of meta-algorithms are aimed at combining weak learners [93]:

1. In an approach called bagging, the weakest learners are learned independently from each other in parallel and then combined according to some form of the deterministic averaging process.

2. The boosting approach, which is often used on homogeneous weak learners, learns them sequentially in an adaptive way (each model is dependent on the previous model) and combines them using a deterministic strategy.

3. Stacking, which involves combining weak models that often consider heterogeneous weak, by training a meta-model that outputs a prediction based on the different weak model predictions.

In very general terms, bagging is mainly focused on producing ensemble models with a smaller variance than their components. While boosting and stacking are primarily focused on making more robust models that are less biased than their parts (even if variance can also be reduced).

Bagging Whether we are dealing with a classification problem or a regression one, training a model gives us a function that takes a set of inputs, returns a set of outputs, and is defined concerning these inputs. The training dataset has a theoretically higher variance than the actual training dataset (As a reminder, a dataset is an observation derived from an even unknown distribution). The fitted model is also susceptible to variability: we would have derived a different model if a different dataset had been observed [94].

The goal of bagging is then straightforward: we want to "average" multiple independent models' predictions to attain a lower variance model. Unfortunately, we can't fit fully independent models in practice since it would require too much data. So we use the excellent properties of bootstrap samples (representativeness and independence) to provide models that are close to independent.

We begin by creating multiple bootstrap samples to act as independent datasets based on actual distributions. After that, we can fit a weak learner for each sample and aggregate the results so they have an "averaged" output. As estimated by the bootstrap samples, the learned base models are approximatively independent and identically distributed (i.i.d.).In such cases, "averaging" weak learners will not alter the expected outcome but reduce its variance [93].

So, assuming that we have n bootstrap samples (n independent datasets) of size B denoted

$$\{Z_1^1, Z_2^1, ..., Z_B^1\}, \{Z_1^2, Z_2^2, ..., Z_B^2\}, ...\{Z_1^n, Z_2^n, ..., Z_B^n\}$$

$$Z_B^n = b - th\ observation\ of\ the\ n - th\ bootstrap\ sample$$

There will be n independent learners, one for each dataset, and they will be aggregated into so called ensemble models with less variance.

$$v_n(.) = \arg\max_k [card(l|w_{(.)} = k)] \tag{3.2}$$

We do the majority vote. For classification, were w_n is the model's output, and v_n is the majority vote for classification [92, 93]. The classification problem can be seen as a voting problem, which means that each model will output a class based on how many votes it receives, and the ensemble model will return the one with the most votes (it is also called hard voting). We also have the option of taking into account the probability of each class returned by all of the models, averaging these probabilities, and keeping the class with the highest probability (soft-voting). Averages or votes can either be weighted or straightforward if any relevant weights can be used, as shown in Figure 3.4.

In conclusion, we might note that bagging has the advantage of parallelism. Different models can be fitted independently from one another, so intensive parallelization techniques can be used if necessary.

After finishing each independent Instructor's training, we count the number of Instructors who voted for each class and then perturbed that result by adding random noise sampled from the Laplace or Gaussian distribution. According to the noisy-max mechanism, when two output classes receive an equal (or quasi equal) number of votes from the Instructors, the noise will ensure that the class with the most votes will be one of these two classes chosen at random. On the other hand, if most of the Instructors agreed on the same class, adding noise to the vote counts will not make a difference in the reality that this class has given the most votes. This delicate arrangement provides accuracy and privacy to our model predictions made by the noisy aggregation mechanism. We can see the architecture of our model in Figure 3.1.

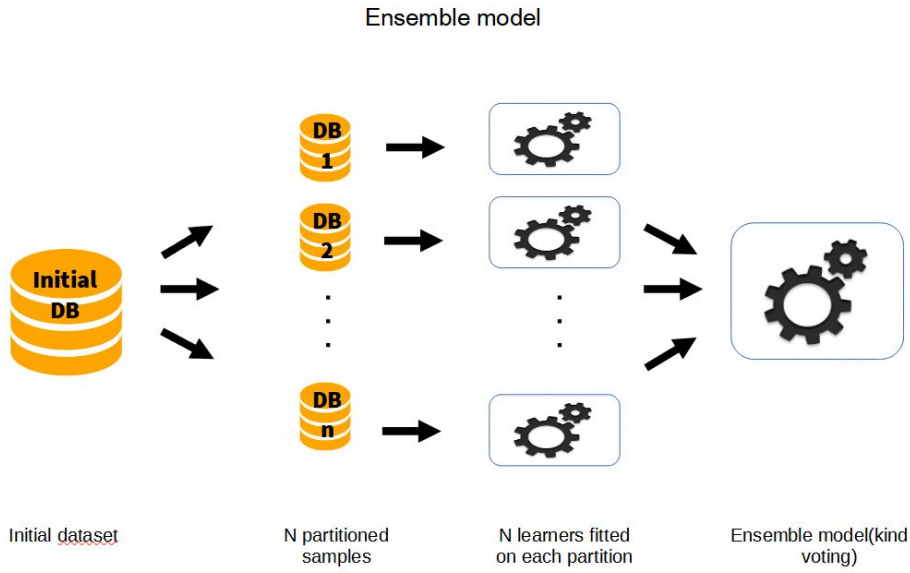


Figure 3.4 Bagging consists in fitting several base models on different partitions samples and build an ensemble model that results of these weak learners.

3.2.2 Differentially Private Learning Algorithms

As we discussed in introduction chapter, one of ML systems' critical elements is the learning algorithm. By protecting the learning algorithm's privacy, we can prevent form model inversion and model extraction attacks. In algorithm 7, we have presented the pseudo-code of the

DPAdam learning algorithm.

Algorithm 7: Differentially Private Adam Algorithm

Result: $\begin{Bmatrix} W \\ \\ b \end{Bmatrix}$

Weight and bias initialization

for *i in range of training steps* **do**

  Batches=miniBatches Generator(X,Y,batch-size)

  for *j in range of batches* **do**

    minibatchesX,minibatchesY=batches[j]

    forward propagation using minibatchX to calculate y$'$

    Calculate cost using minibatchY

    Backward propagation to calculate ;

    ∂W and ∂b

    Clip gradient

    Add Gaussian noise

    Updating the parameters W and b

  end

  return $\begin{Bmatrix} V_{\partial W} = \beta_1 V_{\partial W} + (1-\beta_1)\partial W \\ V_{\partial b} = \beta_1 V_{\partial b} + (1-\beta_1)\partial b \\ V_{\partial W}^{corrected} = \frac{V_{\partial W}}{1-\beta_1^i} \\ V_{\partial b}^{corrected} = \frac{V_{\partial b}}{1-\beta_1^i} \\ S_{\partial W} = \beta_2 S_{\partial W} + (1-\beta_2)\partial W^2 \\ S_{\partial b} = \beta_2 S_{\partial b} + (1-\beta_2)\partial b^2 \\ S_{\partial W}^{corrected} = \frac{S_{\partial W}}{1-\beta_2^i} \\ S_{\partial W}^{corrected} = \frac{S_{\partial b}}{1-\beta_2^i} \\ W = W - \alpha . \frac{V_{\partial W}^{corrected}}{\sqrt{S_{\partial W}^{corrected}}+\epsilon} \\ b = b - \alpha . \frac{V_{\partial b}^{corrected}}{\sqrt{S_{\partial b}^{corrected}}+\epsilon} \end{Bmatrix}$

end

This algorithm outlines our primary method for training a model by minimizing the loss function. The way we used in our work is according to Abadi et al. [13] work. With this difference that instead of SGD, we used Adam. At each step on Adam, we compute the gradient of loss function based on miniBatch size, then we clip l_2 norm of each gradient and calculate its average between this batch size. Then we add noise to protect privacy. We need to compute our proposed mechanism's privacy loss according to our privacy accountant's [13]

information introduced by Abadie's work.

To prove our algorithm's DP guarantee, we require bounding each example's effect on the loss function's gradient. For reaching this target, we bound it by gradient clipping. In the usual way of using a gradient, clipping is used after averaging, but here we do it before averaging because we need to bound each example's effect.

In our work, we have two main difference with the Adam training algorithm

1. Gradient clipping
2. Adding Gaussian noise

Gradient clipping According to the algorithm, at first, we clip the gradient because finding the gradient's optimal value is iterative. In some cases, we may face the problem of the accumulated gradient (gradient expanding). The problem arises when the gradient of the parameter is very large. So the parameters could be thrown very far if a gradient descent parameter update is applied. Thus, this will undo significant work that has been done to come to the current solution [6]. To solve this problem, we can do two things:

1. Gradient scaling

   Scaling involves normalizing a gradient vector to a fixed value, such as 1.

2. Gradient clipping

The reason for gradient clippings is that if the gradient values exceed an expected range, gradient clipping is used to force them to a specific minimum or maximum value.

The gradient clipping heuristic appears when the traditional algorithm does not have a very small step, causing the algorithm to move in the gradient direction rather than remaining outside the region where it indicates the steepest descent [6]. We have two methods for gradient clipping:

1. Gradient clipping by value:

   Clipping-by-value is straightforward. We define a minimum clip value and a maximum clip value.

   We clip the gradient if it exceeds some threshold value; if it is below the lower limit, we clip that as well to the lower limit of the threshold [95].

2. Gradient clipping by norm:

   The clips-by-norm strategy is similar to the clips-by-value procedure. By multiplying the gradients' unit vectors with the threshold, we clip the gradients.

Algorithm 8: Pseudo-code for gradient clipping by norm

Result: g as gradient
if $||g|| \leq threshold$ **then**
  $g \leftarrow \frac{threshold}{\|g\|} g$
end

One of the crucial issues for differentially private Adam is the overall privacy cost of the training set. For measuring the total amount of privacy in our work, we used the moment accountant mechanism which it was proposed by Abadi et al. According to the composition theorem of DP, [12], we can use the accountant procedure to compute the privacy cost for accessing the training set based on our algorithm's iterations [13].

Adding Gaussian noise The second important part in our proposed algorithm is adding Gaussian noise to our gradient in each iteration. lot of research has been done to define the amount of privacy loss for a particular noise distribution [96]. In our work, we have used Gaussian noise and if we initialise the parameter of σ in DPAdam algorithm base on below equation from [13],

$$\sigma = \frac{\sqrt{2 \log \frac{1.25}{\delta}}}{\varepsilon} \tag{3.3}$$

Then based on standard arguments [12], each step is (ε, δ)-differential private with respect to the each miniBatch. As we use miniBatch in our training model and each miniBatch is a random sample from our database, the privacy amplification theorem [97] [98] shows that each step is $(q\varepsilon, q\delta)$-DP concerning the whole database where $q = \frac{M}{N}$, M is the miniBatch size and N is the total size of the database [13].

In our work for measuring the amount of privacy of training model, we used the momentum accountant mechanism which is one of the contribution of Abadi et al. work. By using of moment accountant [13] which is stronger accounting method, our proposed DPAdam algorithm is $O(q\varepsilon\sqrt{T}, \delta) - DP$ with respect to appropriate amount of noise scale and clipping threshold. In this manner, our bound is tighter in two way [13]:

1. It saves a $\sqrt{log(\frac{1}{\delta})}$ factor in ε part.

2. It saves a Tq factor in δ part.

As we expect the δ to be small and $T >> \frac{1}{q}$

A method that uses adaptive learning-rates could have the best results if your input is sparse. Additional benefits include eliminating the need to adjust the learning rate and achieving the best results using the default value. In comparison to the work Abadi et al., they had chose

There are many kind of training algorithms that we could choose and work on their privacy. To make a comparison among them we found that RMSprop provides the Adagrad platform with the tools it requires to deal with its rapidly shrinking learning capacities [99]. However, it differs from Adadelta in using the root mean square of parameter updates instead of the nominator update rule. Compared with RMSprop and Adadelta, Adam adds bias correction and momentum capabilities. Overall, RMSprop, Adam, and Adadelta all perform well under similar circumstances [100]. The study by Kingma et al. [45] indicates that bias-correction allows Adam to slightly outperform RMSprop as gradients become sparser towards the end of optimization. Adam seems to be the best overall choice [99].

Recently, many papers have used simple Learning Rate Algorithms without momentum and SGD without momentum. The SGD usually finds a minimum. However, it may take much longer than other optimizers, revealing the need for a robust initialization and annealing schedule and appearing in saddle points instead of local minima. Therefore, if you are concerned with fast convergence and intend to train a DNN, you should rely on adaptive learning rate methods.

The reasons for using Adam optimizer is as below [45]:

1. Straightforward to implement. Computationally efficient.
2. Little memory requirements. Invariant to diagonal re-scale of the gradients.
3. Well suited for problems that are large in terms of data and/or parameters.
4. Appropriate for non-stationary objectives.
5. Appropriate for problems with very noisy/or sparse gradients.
6. Hyper-parameters have intuitive interpretation and typically require little tuning.

We can design our DNN for the training phase with different approaches. The architecture of each one of them be different from each other, or we can design it with the same architecture

with different activation functions because we want to make it harder for the adversarial attacker to reach our information by reverse engineering [22, 25, 25, 52, 101–106]. Here at this step, we have designed it with the same architecture but a different dataset for the training of the network.

From the beginning of our approach until this part, we have preserved our information's privacy, and now we want to go one step forward.

3.3 Phase 3. API Hardening

We pre-processed our data to protect our information's privacy, then using the ensemble technique to increase our prediction accuracy and use DPAdam to protect our parameters and our model's privacy. The process diagram of this phase is shown in Figure 3.1 in the API hardening phase. We are no longer able to publish an ensemble of instructor models publicly. Otherwise, an adversary might examine the published instructors' internal parameters to discover private data that they trained on.

One critical part of ML systems is the prediction layer. Most of the time, the attackers have access to this part of the system directly, and they can send a query and get a response. By sending queries and responses, they can access the model, and if they access the model, they would be able to access training data. To protect the system and preserve the system's privacy, we add an extra phase in our architecture. We can make our system that includes the ensemble training and protected training set by this work. In this part of the system, we use the privacy-protected labels and the ensemble model's aggregation to produce the final output. The new training step which we have here use also the differential private training model.

We train an "understudy" on nonsensitive and protected data, labeled using an aggregation mechanism using the instructor's training. The "understudy" model is used in place of the "Instructor" ensemble so that the privacy loss is fixed to a value that does not increase with the number of user queries made to the understudy model.

Indeed, the amount of privacy loss is determined by the number of queries to the "instructor" ensemble during "understudy" training and is not correlated with the number of queries to the deployed "understudy" model.

Thus, even if such an understudy's architecture and parameters are publicly known or are reverse-engineered by an adversary, users' privacy who contributed to the original training dataset is preserved.

Our proposed architecture includes an extra phase that increases our model's privacy and

training data, and it is deployed for users to answer any prediction queries. After this point, the private data and ensemble of instructor models can be safely discarded since only the understudy model is used to inference the prediction layer. The overall privacy budget is fixed to a constant value once the understudy has completed its training. Second, an adversary with access to the understudy's internal parameters could, in the worst-case, only recover the labels on which the understudy trained, which are private, and the rest of the system would be protected.

Based on the model of our work as we see in Figure 3.1 we preserve the privacy of output by using Gaussian differential private mechanism and then send the result to a next layer DNN. As we call it the "Understudy" phase in the new training step, we have used the same architecture as we had in the previous layer, but it is also possible to have a different DNN. It will also preserve the robustness of our system against attacks [22, 25, 25, 52, 101–106] and by and finally, we publish the result of this phase as privacy protected output of our work. In the next chapter, we will discuss and analyze the results.

3.4 Evaluation Methodology

3.4.1 Evaluation of Pre-Processing (Phase I)

With noise addition, it is important that all data be transformed in such a way that their statistical properties are the same that they were original. Statistical characteristics such as normal distribution, mean, variance, standard deviation, covariance, and correlation must be considered for primary and differential private data sets.

It is important to measure how related the original data and the perturbed data are with noise addition. Covariance, Cov(X, Y), calculates how related the deviations between the data points X and Y are. If the covariance is positive, then the X and Y data points tend to increase together. If the covariance is negative, then the tendency is that one loses for the two data points X and Y while the other gains. However, if the covariance is zero, this illustrates that the data points are each independent [76].

The other important data measurement is correlation r_{xy}, which is also known as the Pearson product that calculates the capability and inclination of an additive or linear relation between two data points. If the correlation metric can be between $[-1, 1]$ if the correlation metric is -1, it will show a negative linear relation. If it is 0, it will show there is no relation between them. If it is near 1, it means they have a linear connection between each other [76].

we calculate the correlation between two images according to formula:

$$r = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \bar{A})^2)(\sum_m \sum_n (B_{mn} - \bar{B})^2)}} \tag{3.4}$$

Where $\bar{A}$=mean(A) and $\bar{B}$= mean(B) and m, n are dimension of image. Here we calculate the correlation between each protected and non-protected entry in our dataset. Then we get the correlation average between the primary dataset and the differential private dataset. The correlation average of our dataset is 0.81254, and it will show that two data sets, increasing and decreasing together because it is near to 1.

By this phase of privacy preservation, we also have prevented our training model from the over-fitting problem.

3.4.2 Evaluation of Ensemble Learning (Phase II)

With the ensemble technique, we try to increase the privacy of the training dataset and learning model. If we use just one ensemble, as the number of training datasets will increase, the model generalization and performance will improve. When we use ten ensembles, we will increase the privacy ten times more than the system with one ensemble. For evaluation of the ML system, we used accuracy, F-score, precision, and recall. According to the proposed DPAdam, we can have a different amount of privacy for the privacy of design. But we should set a trade-off between privacy and accuracy of our system, and we have expanded them in detail in last chapter.

3.4.3 Evaluation of API Hardening (Phase III)

For evaluating the output of our ML system, we measured the accuracy, F-score, precision, and recall of system with protected label and non-protected label and also measured the privacy guarantee of system. We have explained all the details with charts and diagrams in last chapter.

3.5 Implementation

We have implemented our architecture and algorithm with Python language in PyTorch. We have some main functions in our implementation as below:

1. Pre-processing of training dataset and make a protected dataset:

According to the amount of Gaussian noise which we add to our data with $mean = 0$ and $\sigma = 0.6$, it preserves the privacy of $(2.5, 10^{-5})$-DP.

2. Partitioning and defining the datasets for Instructors:

   We divide the training dataset into the number of Instructors and call them Instructor partitioned dataset. Then load them with different data-loaders.

3. Define the Understudy dataset and separate it into training and testing datasets:

   Here, we used the MNIST test-set for the Understudy phase, and we divide it into two parts. Understudy training data and Understudy test data. We used most of the data for the training set with a ratio of $\frac{9}{10}$ for training and $\frac{1}{10}$ for the test-set. And load them into the data-loader.

4. Defining a Neural Network Model:

   For defining the neural network model, we used three different implementations,

   (a) The first model is a simple CNN network with 2 convolution layer and 2 fully-connected layer.

   (b) The second model is Alex-net with 5 convolution layer and 2 fully-connected layer

   (c) The Third model is a VGG network with 13 convolution layer and 5 max-pooling layer and 3 fully-connected layer.

3.5.1 Differential private CNN implementation

We implemented our differential private model on a deep convolutional neural network (CNN) according to three different architecture: the simple two-layer CNN, the Alex-net, and the VGG-net. We got the classified output for the implementation. In this section, we explain the VGG-net implementation. It has a giant network with more layers. The two other deep network architectures are the same, with less convolutional layers, and they are more straightforward than VGG-net.

We have used PyTorch for our implementation. It has its data structure, which is Tensor, and our data set is also MNIST. The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It contains a 60000 small 28×28 pixel square image of handwritten base digits from 0 to 9. The reason for choosing this dataset is because it is a widely used and deeply understood dataset. They use it as a baseline for scientists to compare their implementation

methods with each other. The highest classification accuracy rate for MNIST is above 99%.

Figure 3.5 shows a sample of the MNIST dataset with different amounts of noise we add to it to preserve data privacy. When we have real data with no privacy protection, the data is pure, and the utility and accuracy of learning tasks are high. As we increase the privacy protection, the utility of learning tasks will decrease. The model can not extract too many features to produce a more accurate prediction.

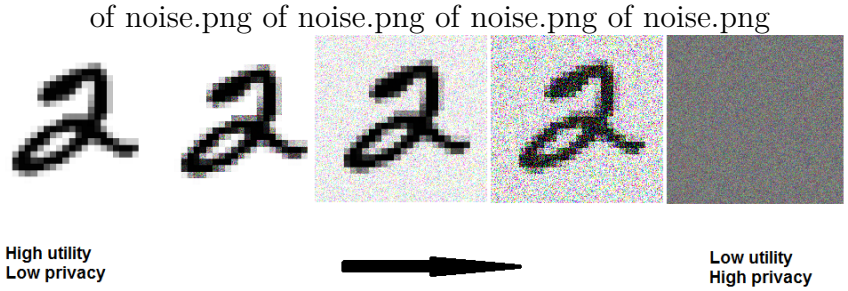


Figure 3.5 This a sample of MNIST dataset,from left to right as much as we increase the privacy (increasing the amount of noise) we will lose accuracy. The first image is with $\varepsilon = \infty$ (no privacy) and the right image is with $\varepsilon = 0^{+}$ (ultimate privacy). we should set a trade-off between the privacy dataset and accuracy of our ML system.

Our estimate of the true mean is also very close to the true mean in the case of 10,000 data points. This means we can add much noise as long as there are many data points and still maintain the accuracy and reliability of reported data. Hence, Local Differential Privacy works better when we have more data. This might sound counter-intuitive, but it makes sense as more data tends to cancel out the noises and provide a more accurate representation of the population.

In our work, for estimating the general performance of our model on MNIST classification. We can split the whole process of a DNN into four separate steps. We explain three first steps here and then in 4th chapter we explain the presentation of the results and discuss them :

(a) Loading the pre-processed dataset.

(b) Preparation of dataset.

(c) Definition of the model.

(d) Evaluation of the model and result presentation.

Loading the pre-processed dataset We know that each image is a differentially private image because we have done the pre-processing phase, and now we have a privacy preserved dataset. Each of these images is a handwritten digit, and all of them are the same size (28×28) and gray-scale. Here we have 10 classes for each number to classify using a one-hot encoding vector for the classes to transform the integer into a ten element binary vector with a 1 for the index of the class value and 0 values for all other classes.

Preparation of database As we know, we are using a gray-scale image dataset; the pixel value is between 0 and 255 (black to white). Here we normalize the pixel values and re-scale them in range of $[0, 1]$. By this normalization, we convert the data type from unsigned integer to float. The other preparation step is adding Gaussian noise to the data according to the amount of privacy we want. As much as we increase the amount of noise, our privacy will increase, but the model's performance will decrease. Our images in MNIST dataset are in one channel, and for feeding the data in VGG-net, we should change them to three-channel images and change the scale of images to 256×256.

Definition of the model Now for defining the model we can separate it to two parts:

(a) Feature extraction

   i. Convolutional layer

   ii. pooling layer

(b) Classifier

We use a convolutional layer with a filter size of (3) and a modest number of filters (32) followed by a max-pooling layer. We use the filter maps for providing the features for the classifier.

As this problem is a multi-level classification, we need an output layer with ten classes, and for the output layer, we use the softmax activation function. For the dense layer between output and feature extractor layer, we use 100 nodes for the dense layer, and the activation function for these dense layers is the ReLU activation function. In defining the model process, we use our proposed DPAdam optimizer in our model.

This model is a low deep model, and we can increase the depth of our model. With the VGG-net model's help, we increase the feature extraction part's depth by adding more convolutional and pooling layers with the same sized filter while increasing the

number of filters. We added a double convolutional layer with 64 filters each, followed by another max-pooling layer. We have implemented three various deep architectures, a two-layer CNN network, Alex-net, and VGG-net, to compare them in the evaluation part for our implementation part.

5. Training the Instructors' datasets and finding the Instructor models.

6. Using all the Instructor models, find predictions for the Understudy training dataset.

   We define a function as **predict** to find each Instructor's prediction and return the prediction as an array. These are the predictions that we get when we run the understudy training-set on each Instructor model. So the shape of the prediction array is [The number of Instructor, The number of Understudy training set].

7. Getting the private label for the Understudy training dataset.

   After that, according to every prediction, we get the private training label for the Understudy training set. In this way, we preserve data privacy in the border layer, which is our Understudy layer, as we use it for prediction (API prediction) hardening.

8. Replacing the old labels from the understudy training dataset with private labels.

   After that, we remove the labels from the Understudy train-loader, and we add the private labels, respectively.

9. Training the Understudy training dataset with new labels and testing accuracy with the Understudy test dataset.

CHAPTER 4 RESULTS

In this chapter, we discuss and represent the results of our work. At the beginning, to test the statistical features of differentially private datasets, we used a random generated dataset to visualize the effect of noise on them and analyse their difference according to correlation metric and the accuracy of model with different amount of privacy. Then, we used the MNIST dataset which is one of the well known dataset for image classification task and compare our work with the other works in this field. also we implemented three different CNN networks (two-layer CNN, Alex-net, and VGG-net) model to test our work.

4.1 Generated dataset results

With the intent to test how the statistical features of data change when we add noise for preserving privacy and how the ML system overcomes this defect following the method described in this section . To this end, we used our synthesize DILL dataset. We develop a small model utilizing the Keras deep learning library [107] for the problem. This model has two inputs, 25 hidden nodes, and one output class. We used the ReLU activation function for the nodes in the hidden layer. As the problem is a binary classification, we used the sigmoid activation function, which is a desirable one for the output layer, to predict the probability of a sample belonging to class A or B. These result while we do not have privacy in our system, and we can see it in Table 4.1. We can attempt to fit models with various training datasets and evaluate how they perform on a test set.

Too few examples will result in poor test performance because the model will over-fit the training set, or the training set will not represent the problem.

When we increase the training set's size, it will result in a good performance, but perhaps slightly lower than ideal test accuracy. Because the chosen model cannot learn the subtleties involved with such a large training dataset, or the dataset is over-representative of the problem.

After checking our model's output with no protected data, we used a different amount of Gaussian noise, with mean=0 and σ in the range $[0, 10]$ to have a different amount of privacy in our dataset. We can see the result of a deep network in table 4.2 for $\varepsilon = 0.2$ and in table 4.3 for $\varepsilon = 0.4$.

In Figure 4.2 we can see the accuracy of our model with $\sigma = 0.1$ over the number of instances in our generated dataset. As we can see, when we increase the number of instances in our

Accuracy of non-protected model with generated dataset	
Train size	Test Accuracy
100	93.89
1000	94.71
5000	94.87
10000	99.99
100000	99.99

Table 4.1 Accuracy of proposed model without differential privacy.

Accuracy of deep neural network model with protected data with $\varepsilon = 0.2$	
Train size	Test Accuracy
100	74.033
1000	83.668
5000	84.113
10000	83.953
100000	84.953

Table 4.2 Accuracy of deep neural network model with protected data with $\varepsilon = 0.2$.

Accuracy of deep neural network model with protected data with $\varepsilon = 0.4$	
Train size	Test Accuracy
100	84.900
1000	84.800
5000	86.990
10000	86.209
100000	87.426

Table 4.3 Accuracy of deep neural network model with protected data with $\varepsilon = 0.4$.

model, the accuracy of prediction will also increase, and this is the statistical feature of the deep network(as much as we increase the number of the training dataset, the accuracy of classification will increase).

We can show the data's spread with a box and whisker diagram (or box plot). This diagram shows the quartiles of the data, using these as an indication of the spread.

This diagram is made up of a "box", which lies between the upper and lower quartiles. The median was indicated by dividing the box into two and we can see the distribution of output answers as we can see in the Figure 4.2.

The performance of neural network models will improve as more and more data is supplied to the model. Eventually, there will be diminishing returns where more data will not clarify how

(1).png (1).png (1).png (1).png

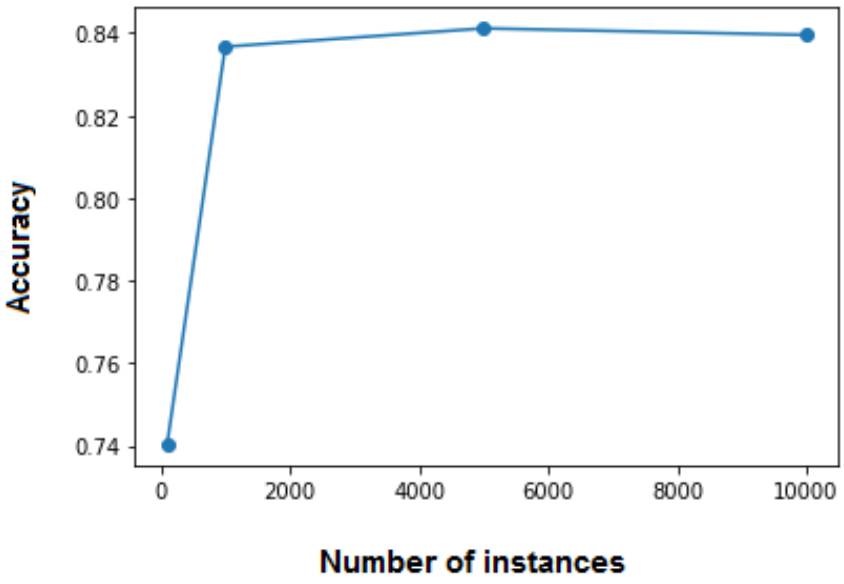


Figure 4.1 Accuracy of the multi-layer-perceptron model with protected training dataset according to number of training dataset.

to model the mapping problem. This point will be reached earlier for more straightforward problems than for more complex issues, such as identifying objects' picture.

4.2 Result presentation for MNIST dataset

We have run our codes in two separate systems. The first one is a GPU server with NVIDIA TITAN V with 12 GB HBM2 dedicated memory for GPU and 640 Tensor Cores with 64 G memory on the server and the other system with the configuration of GeForce GTX 1060 graphics card with 6 Gb dedicated memory for GPU and 16 G memory for the system with a Core i7 10th generation CPU. All the models are with ten instructors in the ensemble model. Just for comparing our architecture's execution time with 1 instructor and 10 instructor, both are run on the same system with the same configuration(TITAN V GPU.)

After implementing the CNN with no privacy mechanism and pure datasets with two-layer CNN, Alex-net, and VGG-net architecture. In Figure 4.3b is shown the confusion matrix for two-layer CNN with no privacy and the loss plot in Figure 4.7b, in Figure 4.3d is shown the confusion matrix for Alex-net with no privacy and the loss plot in figure 4.7d and in Figure 4.3f is shown the confusion matrix for VGG-net with no privacy and the loss plot in Figure

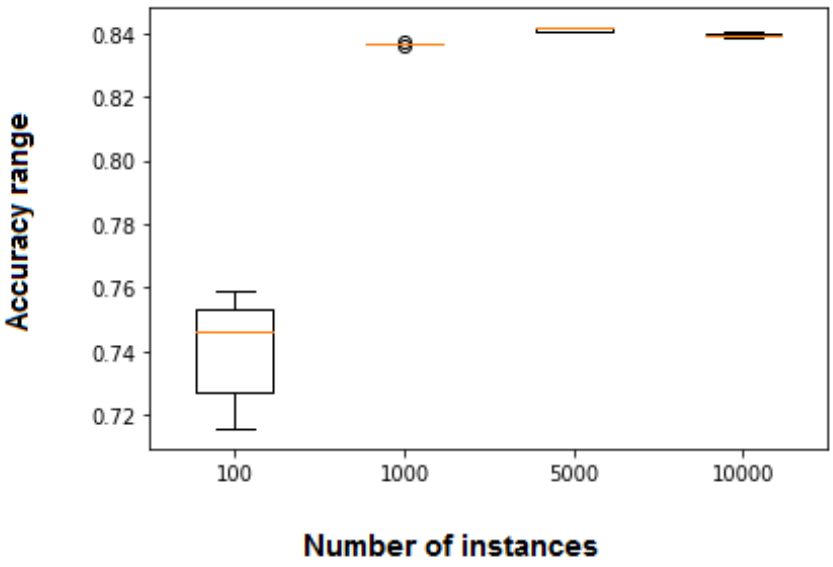


Figure 4.2 Accuracy distribution of the Multi-layer perceptron model with protected training dataset according to number of training dataset.

4.7f. We reach the maximum accuracy of 99%, F-score 99%, Recall 99% and Precision 99% and it is shown in table 4.4.

Comparison of different architecture				
Name of model	Accuracy	F-score	Recall	Precision
Simple CNN	98	97	98	97
Alex-Net	99	98	98	98
VGG-Net	99	99	99	99

Table 4.4 Comparison of accuracy, F-score, recall and precision of different architecture with non-protected dataset.

As the first step of our work, we preserved the privacy of training dataset with $(2.5, 10^{-5})$-DP by using Gaussian noise mechanism. The result of our architectures is shown in table 4.5. In Figure 4.3a is shown the confusion matrix for two-layer CNN with $(2.5, 10^{-5})$-DP of training dataset and the loss plot in Figure 4.7a, in Figure 4.3c is shown the confusion matrix for Alex-net with $(2.5, 10^{-5})$-DP of training dataset and the loss plot in Figure 4.7c and in Figure 4.3e is shown the confusion matrix for VGG-net with $(2.5, 10^{-5})$-DP of training dataset and the loss plot in Figure 4.7e. We reach the maximum accuracy of 98%, F-score 98%, Recall 98% and Precision 98% and it is shown in table 4.5.

Comparison of different architecture				
Name of model	Accuracy	F-score	Recall	Precision
Simple CNN	97	96	97	96
Alex-Net	98	98	96	98
VGG-Net	98	98	98	98

Table 4.5 Comparison of accuracy, F-score, recall and precision of different architecture with just protected dataset after pre-processing phase.

For the second step of our work, we preserved the training dataset's privacy with $(2.5, 10^{-5})$-DP by using Gaussian noise mechanism, and we used our proposed DPAdam training algorithm. The result of our architectures is shown in table 4.6. In Figure 4.5a is shown the confusion matrix for two-layer CNN with $(2.5, 10^{-5})$-DP of training dataset and the loss plot in Figure 4.9a, in Figure 4.5c is shown the confusion matrix for Alex-net with $(2.5, 10^{-5})$-DP of training dataset and the loss plot in Figure 4.9c and in Figure 4.5e is shown the confusion matrix for VGG-net with $(2.5, 10^{-5})$-DP of training dataset and the loss plot in Figure 4.9e. We reach the maximum accuracy of 97%, F-score 97%, Recall 97% and Precision 97% and it is shown in table 4.5.

Comparison of different architecture				
Name of model	Accuracy	F-score	Recall	Precision
DP-Simple CNN	95	95	94	95
DP-Alex-Net	96	96	97	96
DP-VGG-Net	97	97	97	97

Table 4.6 Comparison of different architecture with DPAdam and protected data set Comparison of accuracy, F-score, recall and precision of different architecture with DPAdam training model and protected dataset.

For the third step of our work, we preserved the training dataset's privacy with $(2.5, 10^{-5})$-DP by using Gaussian noise mechanism. We used our proposed DPAdam training algorithm and preserved the API part's privacy with (0.2)-DP by using the Laplacian mechanism. We also used the DPAdam training algorithm for understudy training.We reach the maximum accuracy of 97%, F-score 97%, Recall 96% and Precision 97% as it is shown in table 4.7.

As the VGG-net model is more profound than the two-layer CNN and Alex-net, its performance is more than the two other models. As it is shown in table 4.8, we can see that as much the amount of ε variable increase, the amount of performance will increase because the amount of ε has the inverse relation with the amount of noise which we add for preserving the privacy. As much as we grow the ε, the amount of noise will decrease, and we will have less

Comparison of different architectures				
Name of model	Accuracy	F-score	Recall	Precision
DP-Simple CNN	95	96	95	96
DP-Alex-Net	96	96	96	96
DP-VGG-Net	97	97	96	97

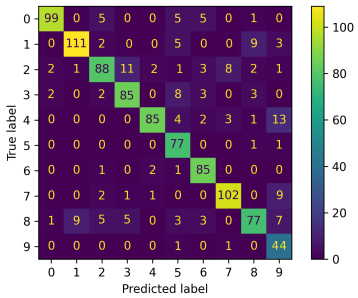
Table 4.7 Comparison of different architecture with DPAdam and protected data set and protected understudy step.

privacy. Unluckily, there is not much agreement about what values of ε are actually "private enough." Most experts accept that values between 0 and 1 are excellent, values above 10 are not good, and values between 1 and 10 are various degrees of "good privacy."

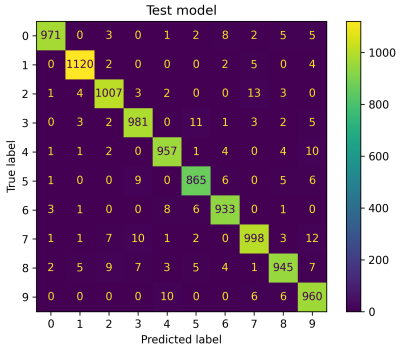
Comparison of different amount of privacy on accuracy of architectures				
Name of model	Accuracy	F-score	Recall	Precision
$(1,10^{-6}) - DP$	90	91	90	90
$(2,10^{-6}) - DP$	93	92	93	92
$(5,10^{-6}) - DP$	95	94	94	95
$(10,10^{-6}) - DP$	98	98	98	98

Table 4.8 Comparison of VGG-architectures with different amount of privacy on our dataset. As we increase the amount of ε, the privacy will decrease and accuracy will increase.

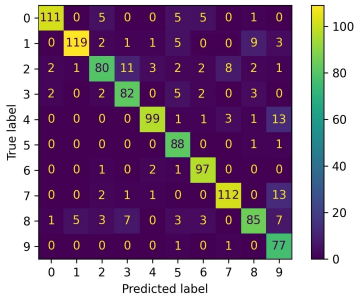
In our architecture, we have partitioned our training dataset to increase the privacy of the ML system. When we increase the number of the training set in a deep network, our model's performance will increase, and the model will train better. We used ensemble technique aggregation to overcome this defect, and even though we have a weaker classifier than our whole training set one model. Comparing one instructor to ten instructors, we will see a 1% increase in final accuracy. But on the other hand, we will have a 10 time less privacy-protected system when we have one classifier. In comparison to the execution time, the execution time for 10 separate classifier is 222918.07 second, and the execution time for 1 classifier is 181234.21 the execution time is 1.2 more than 1 classifier in instructor.



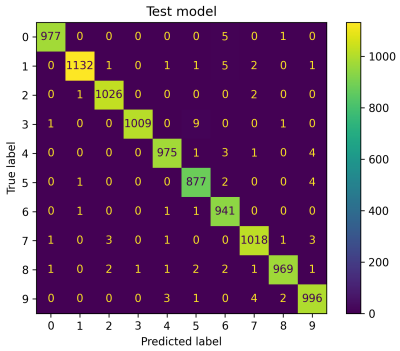
Test confusion matrix for two layer CNN with $(2.5, 10^{-5})$-DP on training dataset and (0.2)DP on output layer



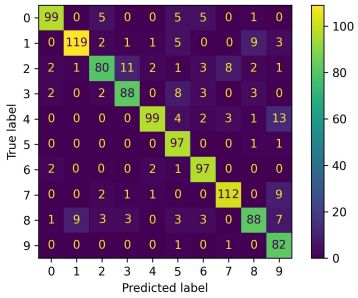
Test confusion matrix with no-privacy for two layer CNN



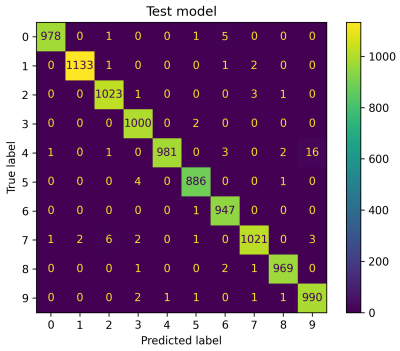
Test confusion matrix for Alex-net with $(2.5, 10^{-5})$-DP on training dataset and (0.2)-DP on output layer



Test confusion matrix with no-privacy for Alex-net

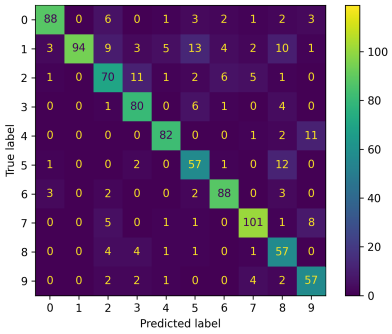


Test confusion matrix for VGG-net with $(2.5, 10^{-5})$-DP on training dataset and (0.2)-DP on output layer

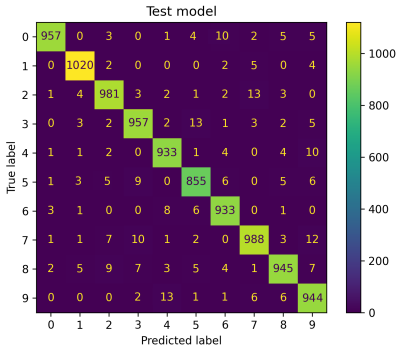


Test confusion matrix with no-privacy for VGG-net

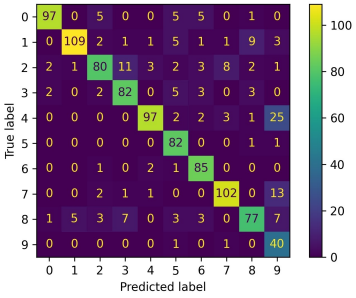
Figure 4.4 Comparison of 2-layer CNN, Alex-net and VGG-net architecture convulsion matrix with no differential private algorithms and $(2.5, 10^{-5})$DP on training dataset and (0.2)DP on output layer



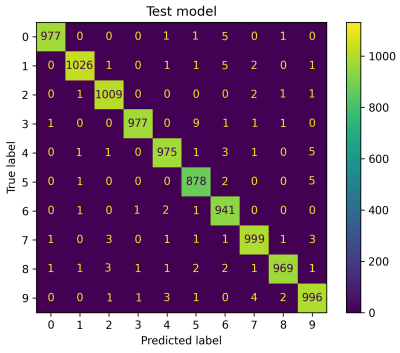
Test confusion matrix for two layer CNN with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm and (0.2)-DP on output layer



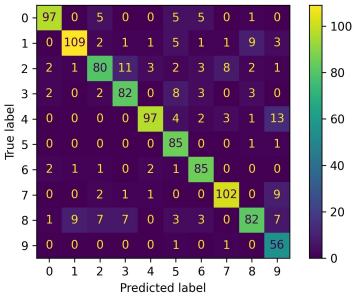
Test confusion matrix with $(7, 10^{-5})$-DP for DPAdam training algorithm for two layer CNN



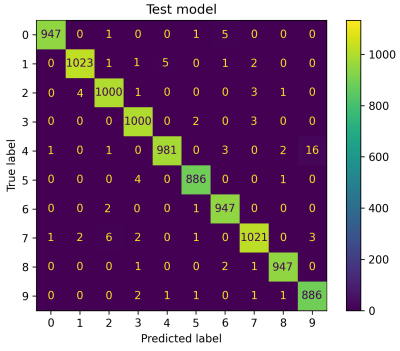
Test confusion matrix for Alex-net with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm and (0.2)-DP on output layer



Test confusion matrix with $(7, 10^{-5})$-DP for DPAdam training algorithm for Alex-net

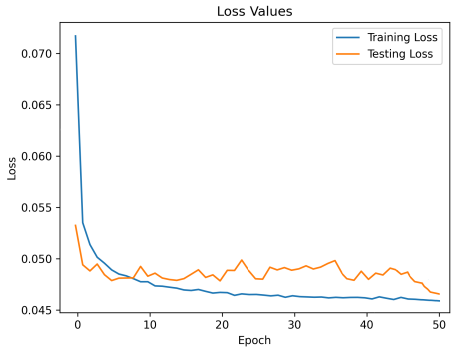


Test confusion matrix for VGG-net with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm and (0.2)DP on output layer

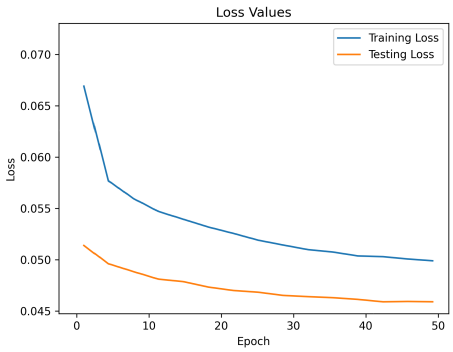


Test confusion matrix with $(7, 10^{-5})$-DP for DPAdam training algorithm for VGG-net

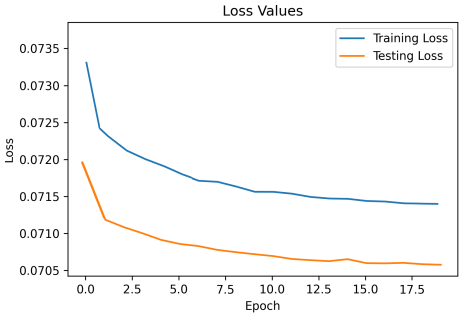
Figure 4.6 Comparison of two-layer CNN, Alex-net and VGG-net architecture convulsion matrix with DPAdam learning algorithms with $(7, 10^{-5})$-DP.



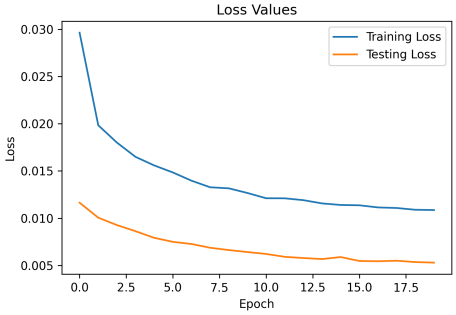
Train and test loss for two layer CNN with $(2.5, 10^{-5})$-DP on training dataset and (0.2)DP on understudy layer



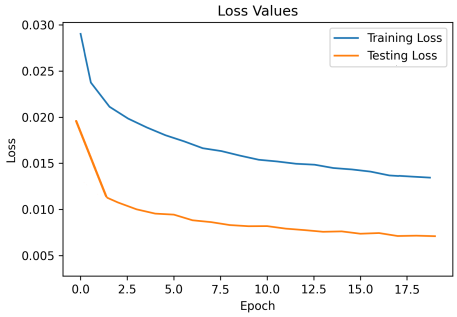
Train and test loss for with no-privacy for two layer CNN



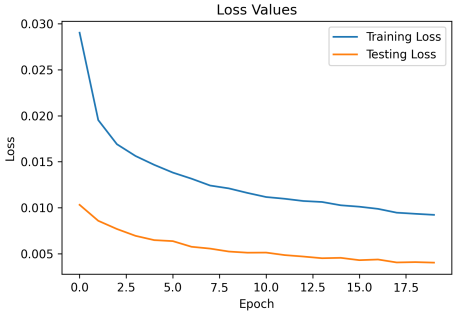
Test confusion matrix for Alex-net with $(2.5, 10^{-5})$-DP on training dataset and (0.2)-DP on understudy layer



Test and train loss with no-privacy for Alex-net

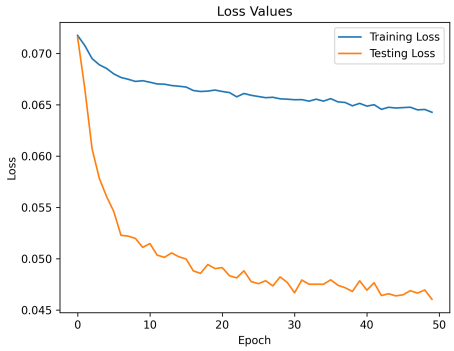


Test confusion matrix for VGG-net with $(2.5, 10^{-5})$-DP on training dataset and (0.2)-DP on understudy layer

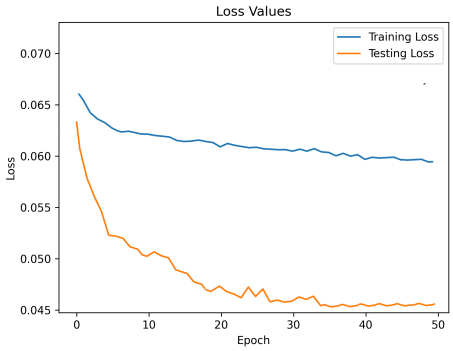


Test and train loss with no-privacy for VGG-net

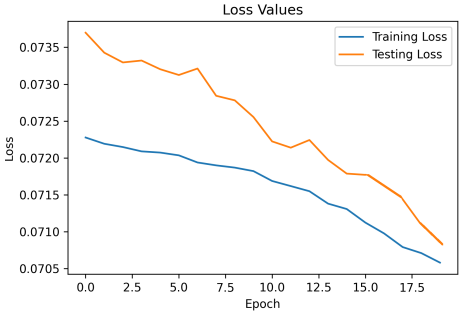
Figure 4.8 Comparison of two-layer CNN, Alex-net and VGG-net architecture loss of architecture with no differential private algorithms.



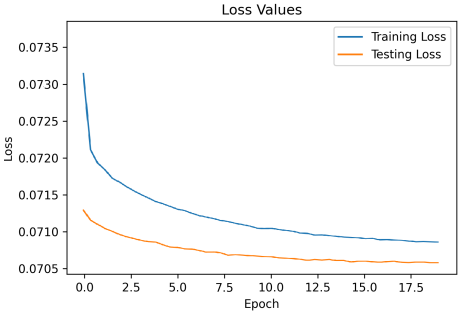
The loss plot for two layer CNN with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm and (0.2)-DP on understudy layer.



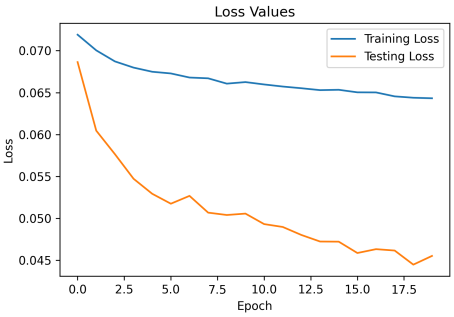
The loss plot for two-layer CNN with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm no privacy on understudy layer.



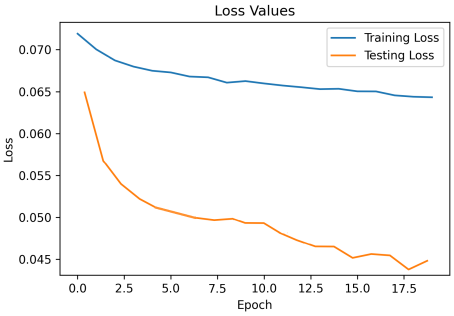
The loss plot for Alex-net with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm and (0.2)-DP on understudy layer.



The loss plot for Alex-net with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm no privacy on understudy layer.



The loss plot for VGG-net with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm and (0.2)-DP on output layer.



The loss plot for VGG-net with $(2.5, 10^{-5})$-DP on training dataset and $(7, 10^{-5})$-DP for DPAdam training algorithm no privacy on understudy layer.

Figure 4.10 Comparison of two-layer CNN, Alex-net and VGG-net architecture loss of architecture with DPAdam learning algorithms.

CHAPTER 5 GENERAL DISCUSSION

The general methodology in this study focused on two main objectives. This research led to a new approach in privacy preservation in ML systems to the best of our knowledge. According to our research, developing an ML system as a crucial part of all modern applications brings many human facilities. But besides all these capabilities, it is a good host vulnerability for attackers to exploit. The two main targets of attackers in the Ml system to exploit are sensitive data and training models. Attackers, by reaching these elements, will extract the information of people or models. Preserving the privacy of both of them is crucial. Because if the attacker penetrates the system and reaches the data, they will extract all the information or access to the model. They would be able to steal models, make reverse model attacks, and extract sensitive data. In the pipeline of the Ml system, these two essential parts are accessible from the training dataset, model, and the prediction layer of the ML system, which is the most accessible part of ML systems. So, according to the above facts, we decided to propose an architecture to overcome these defects and improve the privacy of Ml systems. For the first step according to our architecture, the other works like Mivule et al. used the anonymization and adding noise technique to improve privacy. The anonymization technique is weak, and by having auxiliary information, attackers can retrieve the data. We used differential privacy which is better than the method because by producing randomness in the results, attackers cannot determine if their extracted data is actual or not (the plausible deniability). Besides that, in the second step, according to the previous works in this area, they had proposed methodologies according to a combination of differential privacy with stochastic gradient descent algorithm to improve the confidentiality of the training model. This method has many advantages, but as they had used SGD mode, it had some problems like the high probability of sticking in local minima and its low convergence speed toward the optimal solution compared to modern methods. To overcome this fact and have all the proposed method's advantages, we decided to choose the Adaptive moment(Adam) learning algorithm. The only weak point of this learning algorithm is its' generalization problem. As we add a Gaussian mechanism to preserve the privacy of the learning algorithm, it will act as a penalty therm and regularize the final output. It will prevent generalization and over-fitting problems. So in our proposed model (DPAdam) we could preserve the privacy and generalization problem of Adam learning model.

In the third step, to prevent attacks on the prediction layer, we added a new training step to make the rest of the system a black box and directly control the model's seizures. In the worst case, the attacker can reveal the information of the extra learning step.

In conclusion, our proposed deep learning model obtained good results by using VGG-net deep network architecture. As this CNN network has 19 latent layer, it could retrieve feature from privacy protected training dataset. Generally our proposed privacy-protected training models shows that we can change the privacy-preserving mechanism to a non-zero-sum game and have a good trade-off between privacy and performance.

CHAPTER 6 CONCLUSION

We tried to preserved the privacy of DNN by proposing DPAdam training model and also by proposing an architecture for Ml systems, we tried to preserve the privacy of total system according to existence gaps in pipeline of Ml systems (sensitive training data, training model and also prediction layer of system.) In our experiments on MNIST, we achieve an accuracy of 97% for training with (7, 10^{-5})-differential privacy for training algorithm and (2.5, 10^{-5})-differential privacy for training dataset. Our proposed algorithms are based on a differentially private version of Adaptive momentum (Adam); they run on the PyTorch [108] and Keras [107] software library for machine learning. In comparison, Hamm et al. guarantee privacy only conditionally for the restricted class of students classifier [34], limiting the model to logistic regression with convex loss and compared to Abadi et al., which presents new work in DP of deep learning [13]. They had some primary assumptions about batch selection, loss function, and specific optimization algorithm of stochastic gradient descent (SGD). In our work, we proposed architecture in three distinct phases. In this architecture, in the first phase, we preserve the training dataset's privacy, and we train our model with the protected dataset. We have used the DPAdam training algorithm in the second phase because it converges faster than the DPSGD algorithm. The other reason is that DPSGD has the problems with being stuck in a local minimum, and with DPAdam, we can overcome this defect. Finally, in the last phase of our work, by training a new model with protected training set labels and a by using the DP learning algorithm, we preserve this step's privacy, and we can discard the two previous phases of our system and make them as a black-box system.

In table 6.1, we compare our results with the other works done in this area.

Results on MNIST					
	No-privacy with Adam	No-privacy with SGD	Shokri et.al	DPSGD	DPAdam
(ϵ, δ)-differential private	None	None	report ϵ per parameter	$(10, 10^{-5})$-DP	$(10, 10^{-5})$-DP
Total accuracy	99	98	80	97	97.3

Table 6.1 Comparison of our work with the other similar work

As our work is more similar to Abadi et al., the more detailed comparison with their work is in the table 6.2.

Results on MNIST						
	DPSGD			DPAdam		
differential private-CNN	$(7, 10^{-5})$	$(5, 10^{-5})$	$(2, 10^{-5})$	$(7, 10^{-5})$	$(5, 10^{-5})$	$(2, 10^{-5})$
accuracy	97	94	91	97.3	95	91.7

Table 6.2 Comparison of our work with Abadi et.al work in more detail

With the help of DP, we can:

1. protects our systems against arbitrary risks, and it is beyond protection against re-identification.
2. Including all information about past, present, and future datasets, with DP, we can neutralize the linkage attack.
3. As privacy loss is a quantifiable measure, and we can compare different techniques based on their privacy loss.

6.1 Summary of Works

Machine learning is phenomenal, and nowadays, we can all agree that soon ML will be a standard component of every tech product that anyone uses in daily necessary activities. However, the challenge is that with all the splendid abilities ML provides for us, a host of vulnerabilities is available for attackers to exploit and extract information. By using system vulnerabilities, attacks can perturb your model's thinking and get it to output an incorrect prediction, the same as evasion and poisoning attacks. The other models of attacks allow the attacker to extract sensitive information from your model, or your training and testing data, or the model itself. The other reason for the importance of privacy issue is that big companies like Facebook, Google, etc., have over their users' data as the basis for their power, and possibly for the way they abuse it. In other words, information capitalism has turned data into a means of social control, and privacy harms can trigger a wealth of related offend, chilling freedom of speech, consciousness, communities, etc.

To reach this goal, prevent such attacks on the machine learning algorithm, and protect the training dataset's privacy and our learning model, we proposed an architecture that will make our system like an API base system to prevent API attacks. With the help of DP, we protected the training set's privacy, which we feed our model for training and testing. Also, by introducing a new training algorithm based on DP and Adam optimizer, as we called it DPAdam, we prevented our model's privacy, which is the other significant concern of machine

learning systems. We protected our model from the malicious activity of attackers. Finally, we test our model on the MNIST dataset. According to our results, our proposed method and model have the right balance between privacy and the system's performance.

6.2 Limitations

Although the results presented in this thesis are encouraging, there are some limitations as well. The first constraint is the limited data samples and lack of extensive and independent testing datasets, which is crucial in deep learning. As much as our training set increases, the generalization of our model will increase. The second constraint for running my codes as we used a deep neural network is that it needs a considerable amount of hardware resources, and if I had a GPU server, I could train and test more different models. The third constraint was making a trade-off between privacy and the performance of the model. As much as we increase the amount of privacy of information, the learning algorithm will get far away from the accurate data features, and it will cause our model not to train well. Therefore we had to set a balance between privacy and accuracy. The final constraint is the deep learning models' dependency on the quality and amount of training data, affecting over-fitting.

6.3 Future Research

We present significant recommendations and ideas that give the primary lines and key research inquiries for future work regarding future perspectives.

- Tune pixel scaling

  When we are working with pixel-based (raster images) for our convolutional network, it is crucial to understand that scaling an image for feeding our network rather than stretches images larger or scales them smaller. When scaling, the resolution is not adjusted to best suit the new size, and somewhat the pixels are stretched and appear pixelated. When an image is scaled, its pixel information is changed. The most common side effect of scaling an image larger than its original dimensions is that it may appear to be very fuzzy or pixelated. Scaling images smaller than the actual dimensions do not affect quality as much but can have other side effects. So by improving the re-scaling technique, the accuracy of prediction will get better.

- Tune the DPAdam optimization algorithm

  Finding the best tuning parameter can be done in different ways, but we can categorize most of them into two categories: pre-defining the evaluation values like grid search,

or we can incrementally determine the values. In some cases, the model will have more than one tuning parameter, and we can have a multidimensional parameter combination to choose from. Then we can calculate the results and picked the parameters based on the best empirical result, but the other method can also be selected.

- Tune model depth and using the other architecture of deep networks

  In recent years according to the rapid improvement of deep learning models, we see new architectures and designs with better performance and accuracy. In our work, we used two layers CNN, VGG-net, and Alex-net, for our deep CNN, but the other architecture of CNN, like Res-net, google-net, etc., can be used.

- Tune the design of the proposed model

  In this system design, which we used in our work, we followed multi-purpose to prevent API attack, second, protecting training and test set privacy. Third, protect the machine learning model. Indeed, the other design with different approaches can be used in their work for a different purpose.

- Tune the differential private amount of noise

  In our work, we used (ϵ, δ)-differential privacy. However, the other model is like Rényi Differential Privacy and different calibrated noise distribution models.

REFERENCES

[1] "A brief introduction to differential privacy," https://medium.com/georgian-impact-blog/a-brief-introduction-to-differential-privacy-eacf8722283b.

[2] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[3] "Accuracy of noisy counting," https://desfontain.es/privacy/privacy-loss-random-variable.html.

[4] "Accuracy of noisy counting," https://georgianpartners.shinyapps.io/interactive_counting/, accessed: 2019-08-06.

[5] "Global vs local differential privacy," https://medium.com/@shaistha24/global-vs-local-differential-privacy-56b45eb22168.

[6] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. MIT press Massachusetts, USA:, 2017, vol. 1.

[7] M. Rigaki and S. Garcia, "A survey of privacy attacks in machine learning," *arXiv preprint arXiv:2007.07646*, 2020.

[8] L. Sweeney, "Weaving technology and policy together to maintain confidentiality," *The Journal of Law, Medicine & Ethics*, vol. 25, no. 2-3, pp. 98–110, 1997.

[9] B. Alipanahi *et al.*, "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning," *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[10] A. Kannan *et al.*, "Smart reply: Automated response suggestion for email," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 955–964.

[11] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[12] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[13] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 308–318.

[14] N. Sun *et al.*, "Data-driven cybersecurity incident prediction: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1744–1772, 2018.

[15] S. Samonas and D. Coss, "The cia strikes back: Redefining confidentiality, integrity and availability in security." *Journal of Information System Security*, vol. 10, no. 3, 2014.

[16] J. Andress, *The basics of information security: understanding the fundamentals of InfoSec in theory and practice.* Syngress, 2014.

[17] J. E. Boritz, "Is practitioners' views on core concepts of information integrity," *International Journal of Accounting Information Systems*, vol. 6, no. 4, pp. 260–279, 2005.

[18] G. Loukas and G. Öke, "Protection against denial of service attacks: A survey," *The Computer Journal*, vol. 53, no. 7, pp. 1020–1037, 2010.

[19] L. Muñoz-González *et al.*, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 27–38.

[20] "How to attack Machine Learning (Evasion, Poisoning, Inference, Trojans, Backdoors) attack description," https://towardsdatascience.com/how-to-attack-machine-learning-evasion-poisoning-inference-trojans-backdoors-a7cb5832595c, accessed: 2019-08-06.

[21] "Google's artificial brain learns to find cat videos," https://www.wired.com/2012/06/google-x-neural-network/, accessed: 2012-06-26.

[22] R. Shokri *et al.*, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

[23] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[24] F. Tramèr *et al.*, "Stealing machine learning models via prediction apis," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 601–618.

[25] S. J. Oh, B. Schiele, and M. Fritz, "Towards reverse-engineering black-box neural networks," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019, pp. 121–144.

[26] N. Papernot *et al.*, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[27] J. Bennett, S. Lanning *et al.*, "The netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007. New York, 2007, p. 35.

[28] C. Dwork, A. Nikolov, and K. Talwar, "Efficient algorithms for privately releasing marginals via convex relaxations," *Discrete & Computational Geometry*, vol. 53, no. 3, pp. 650–673, 2015.

[29] D. T. Bourgeois, "Information systems for business and beyond," 2018.

[30] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[31] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[32] K. Nissim, S. Raskhodnikova, and A. Smith, "Smooth sensitivity and sampling in private data analysis," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, 2007, pp. 75–84.

[33] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," in *Advances in Neural Information Processing Systems*, 2010, pp. 1876–1884.

[34] J. Hamm, Y. Cao, and M. Belkin, "Learning privately from multiparty data," in *International Conference on Machine Learning*, 2016, pp. 555–563.

[35] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1054–1067.

[36] Y. Lindell, "Secure multiparty computation for privacy preserving data mining," in *Encyclopedia of Data Warehousing and Mining*. IGI global, 2005, pp. 1005–1009.

[37] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[38] C. C. Aggarwal, "On k-anonymity and the curse of dimensionality," in *VLDB*, vol. 5, 2005, pp. 901–909.

[39] R. Chen *et al.*, "Privacy-preserving trajectory data publishing by local suppression," *Information Sciences*, vol. 231, pp. 83–97, 2013.

[40] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Theory of Cryptography Conference*. Springer, 2016, pp. 635–658.

[41] X. Wu *et al.*, "Bolt-on differential privacy for scalable stochastic gradient descent-based analytics," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1307–1322.

[42] F. McSherry and I. Mironov, "Differentially private recommender systems: Building privacy into the netflix prize contenders," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 627–636.

[43] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[44] N. Phan *et al.*, "Differential privacy preservation for deep auto-encoders: an application of human behavior prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[46] N. Phan *et al.*, "Adaptive laplace mechanism: Differential privacy preservation in deep learning," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 385–394.

[47] J. H. Kalivas, "Overview of two-norm (l2) and one-norm (l1) tikhonov regularization variants for full wavelength or sparse spectral multivariate calibration models or maintenance," *Journal of Chemometrics*, vol. 26, no. 6, pp. 218–230, 2012.

[48] "Comparison od l1 and l2 norm," https://www.kaggle.com/residentmario/l1-norms-versus-l2-norms.

[49] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 2007, pp. 94–103.

[50] A. Friedman and A. Schuster, "Data mining with differential privacy," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 493–502.

[51] P. John D. Cook, PhD, "privacy budget," https://www.johndcook.com/blog/2019/10/14/privacy-budget/, May 2019.

[52] P. Jain, M. Gyanchandani, and N. Khare, "Differential privacy: its technological prescriptive using big data," *Journal of Big Data*, vol. 5, no. 1, p. 15, 2018.

[53] J. Lee and D. Kifer, "Concentrated differentially private gradient descent with adaptive per-iteration privacy budget," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1656–1665.

[54] R. J. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *21st International conference on data engineering (ICDE'05)*. IEEE, 2005, pp. 217–228.

[55] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," *arXiv preprint cs/0610105*, 2006.

[56] P. Liu *et al.*, "Local differential privacy for social network publishing," *Neurocomputing*, vol. 391, pp. 273–279, 2020.

[57] "Local and global differential privacy," https://www.accessnow.org/understanding-differential-privacy-matters-digital-rights/.

[58] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[59] B. Marlin, "Missing data problems in machine learning," Ph.D. dissertation, Citeseer, 2008.

[60] B. M. Lake *et al.*, "Building machines that learn and think like people," *Behavioral and brain sciences*, vol. 40, 2017.

[61] J. Yu *et al.*, "A quasi-newton approach to nonsmooth convex optimization problems in machine learning," *Journal of Machine Learning Research*, vol. 11, no. Mar, pp. 1145–1200, 2010.

[62] V. Jain, H. S. Seung, and S. C. Turaga, "Machines that learn to segment images: a crucial technology for connectomics," *Current opinion in neurobiology*, vol. 20, no. 5, pp. 653–666, 2010.

[63] D. Michie *et al.*, "Machine learning," *Neural and Statistical Classification*, vol. 13, no. 1994, pp. 1–298, 1994.

[64] "7 types of neural network activation functions: How to choose?" https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[65] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[66] W. Wong, "deep learning," https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48, 2020.

[67] B. Marr, "deep learning," https://www.forbes.com/sites/bernardmarr/2018/10/01/what-is-deep-learning-ai-a-simple-guide-with-8-practical-examples/#3cfcd7bb8d4b, May 2018.

[68] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[69] P. Y. Simard *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *Icdar*, vol. 3, no. 2003. Citeseer, 2003.

[70] T. N. Sainath *et al.*, "Deep convolutional neural networks for lvcsr," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8614–8618.

[71] X. Zhang *et al.*, "Accelerating very deep convolutional networks for classification and detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, pp. 1943–1955, 2016.

[72] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[73] M. Sternstein, *Barron's AP Statistics: Advanced Placement Test in Statistics,Pages 49 51.*, ser. Barron's AP Statistics. Barron's, 1998. [Online]. Available: https://books.google.ca/books?id=fgdyWzxGnSQC

[74] C. Spatz, *Basic Statistics: Tales of Distributions, Page 68*. Cengage Learning, 2010. [Online]. Available: https://books.google.ca/books?id=chVUEaHOXy8C

[75] M. L. A. of America, *MLA Style Manual and Guide to Scholarly Publishing, Page 95*, ser. MLA STYLE MANUAL AND GUIDE TO SCHOLARLY PUBLISHING. Modern Language Association of America, 2008. [Online]. Available: https://books.google.ca/books?id=aBTuAAAAMAAJ

[76] M. Crawley, *Statistics: An Introduction using R*, ser. Statistics: An Introduction Using R. Wiley, 2005. [Online]. Available: https://books.google.ca/books?id=iu_PZYpjFJ0C

[77] R. Ott and M. Longnecker, *An Introduction to Statistical Methods and Data Analysis, Pages 171 173*, ser. Available 2010 Titles Enhanced Web Assign Series. Cengage Learning, 2008. [Online]. Available: https://books.google.ca/books?id=ka9ClH2fZc4C

[78] W. N. Tavolga, "Signal/noise ratio and the critical band in fishes," *The Journal of the Acoustical Society of America*, vol. 55, no. 6, pp. 1323–1333, 1974.

[79] N. C. C. G. L. N. Y. C. J. B. M. R. R. Yann LeCun, Courant Institute, "Mnist database," http://yann.lecun.com/exdb/mnist/, May 2019.

[80] K. K. Pal and K. Sudeep, "Preprocessing for image classification by convolutional neural networks," in *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2016, pp. 1778–1781.

[81] T. S. Nazaré *et al.*, "Deep convolutional neural networks and noisy images," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2017, pp. 416–424.

[82] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[83] "Law of large number in ml," https://en.wikipedia.org/wiki/Law_of_large_numbers.

[84] "The magic of gaussian noise," https://desfontain.es/privacy/gaussian-noise.html.

[85] T. T. Allen, *Introduction to engineering statistics and six sigma: statistical quality control and design of experiments and systems*. Springer Science & Business Media, 2006.

[86] "Generate pseudo-random numbers in python," https://www.tutorialspoint.com/generate-pseudo-random-numbers-in-python.

[87] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[88] M. Leshno *et al.*, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[89] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural computation*, vol. 7, no. 1, pp. 108–116, 1995.

[90] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[91] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[92] T. G. Dietterich *et al.*, "Ensemble learning," *The handbook of brain theory and neural networks*, vol. 2, pp. 110–125, 2002.

[93] R. Polikar, "Ensemble learning," in *Ensemble machine learning*. Springer, 2012, pp. 1–34.

[94] M. Galar *et al.*, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2011.

[95] J. Zhang *et al.*, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," *arXiv preprint arXiv:1905.11881*, 2019.

[96] C. Dwork *et al.*, "Our data, ourselves: Privacy via distributed noise generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 486–503.

[97] A. Beimel, S. P. Kasiviswanathan, and K. Nissim, "Bounds on the sample complexity for private learning and private data release," in *Theory of Cryptography Conference*. Springer, 2010, pp. 437–454.

[98] M. Bun *et al.*, "Differentially private release and learning of threshold functions," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 2015, pp. 634–649.

[99] "An overview of gradient descent optimization algorithms," https://ruder.io/optimizing-gradient-descent/index.htmlfn19.

[100] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[101] V. S. Congzheng Song, "Auditing data provenance in text-generation models."

[102] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 36–52.

[103] S. Truex *et al.*, "Demystifying membership inference attacks in machine learning as a service," *IEEE Transactions on Services Computing*, 2019.

[104] J. Hayes *et al.*, "Logan: Membership inference attacks against generative models," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 1, pp. 133–152, 2019.

[105] A. Pyrgelis, C. Troncoso, and E. De Cristofaro, "Knock knock, who's there? membership inference on aggregate location data," *arXiv preprint arXiv:1708.06145*, 2017.

[106] A. D. Sarwate and K. Chaudhuri, "Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data," *IEEE signal processing magazine*, vol. 30, no. 5, pp. 86–94, 2013.

[107] "Keras deep learning library," https://keras.io/.

[108] "Pytorch deep learning," https://pytorch.org/.