Differentially Private Machine Learning

Jue Fan (A0221578B)[*], **Xiao Tian (A0220592L)**[*]

National University of Singapore
{jue.fan, xiao.tian}@u.nus.edu

Abstract

Machine learning may involve the use of sensitive data such as clinical records. Such data can leak from trained machine learning models through privacy attacks such as membership inference attacks, resulting in a breach of data owners' privacy. Thus, *differentially private machine learning* techniques emerge to protect privacy by making it unclear from the trained model whether any particular datum is involved in model training (i.e., achieving a desirable privacy objective called *differential privacy*). Instead of inventing new machine learning models from scratch, these techniques make adequate changes to the training mechanism of existing models. In this report, we give a detailed introduction to the field of differentially private machine learning. We first provide necessary backgrounds on the definition, properties and mechanisms of differential privacy. Then we explain in depth how existing machine learning models, including deep learning models, are adapted to achieve differential privacy. We also include a brief discussion of the implications of such techniques.

1 Introduction

In machine learning (ML), a *model owner* fits a statistical *model* to a *dataset* comprising a number of data (i.e., trains the model). The model captures the hidden patterns or trends behind the data so that it can be used to predict the behaviours of new data or future trends when deployed. For example, hospitals can use past patients' data to train an ML model and predict the progression of disease for new patients; finance companies can use market data to predict future trends of stocks. The data used for ML can thus be **sensitive** (e.g., personally identifiable information (PII), disease records) or **valuable** (e.g., business secrets). Therefore, model owners are more than often obligated to ensure the confidentiality of these data and thus privacy of the *data owners*.

The earliest techniques to protect privacy lie in removing the personally identifiable features from the dataset or grouping similar entries into bins (e.g., a feature POSTALCODE can be replaced with DISTRICTNAME such that multiple data with different POSTALCODE now share the same DISTRICTNAME and become indistinguishable). However, the identity of data owners can still be revealed through *linking attacks* [34], where attackers compare the dataset with another available dataset and search for the common information between the two. Therefore, **model owners should not release any version of their training datasets**, even if the datasets have been processed through the aforementioned techniques. When data owners do not trust that model owner will not release or leak their data, they can also adopt *federated learning* [27], where the model owner does not have direct access to the training data.

Unfortunately, even if attackers have no access to the training dataset, they are still able to reconstruct the training data using only the trained ML model through privacy attacks. A well-known privacy attack is *membership inference attack* [33], where attackers use the trained ML model to train an *attack model* that reliably outputs whether an input datum is contained in the training dataset or not. Such attacks entail that privacy should not be treated equivalently as keeping training datasets confidential. Instead, **privacy should be injected algorithmically into the ML model training mechanism** such that attackers cannot "invert" the trained ML model and recover the training data.

It is worth noting that absolute privacy is impossible: some information about the training data must be present in the trained ML model to make it useful and capture the correct patterns. However, it is possible to achieve privacy **with high probability** when model owners use a randomized training mechanism. Therefore, a probabilistic quantification of privacy, ***differential privacy*** (DP) [15], is widely adopted to assess how likely a randomized training mechanism may leak the information about the training data. The core intuition behind DP is that the privacy of a datum is protected if the trained ML models are (probabilistically) indistinguishable when the datum is present or absent in the training dataset. In other words, attackers are uncertain about whether any particular datum is contained in the training dataset or not.

Differentially private machine learning (DP-ML) refers to ML training mechanisms that achieve DP. This is typically done by injecting artificial noises at various stages of existing ML training mechanisms (see Fig. 1) such that it is not certain that the trained ML model is an outcome of data or noises. In this report, we will explain in depth how DP-ML is achieved in various types of existing ML models.

[*]The authors have made equal contribution to this report.

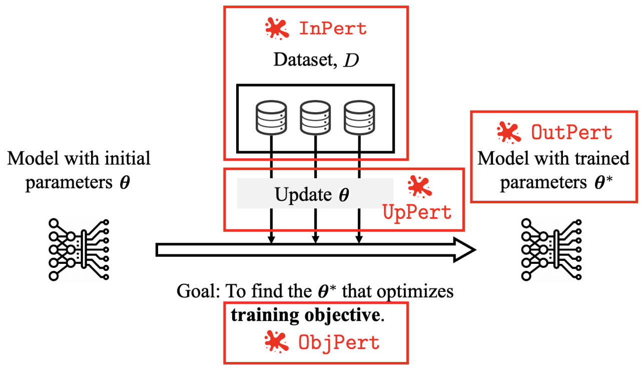

InPert
Dataset, D
Model with initial parameters $\boldsymbol{\theta}$
Update $\boldsymbol{\theta}$
UpPert
OutPert
Model with trained parameters $\boldsymbol{\theta}^*$
Goal: To find the $\boldsymbol{\theta}^*$ that optimizes training objective.
ObjPert

Figure 1: **Overview of DP-ML mechanisms**. ML aims to update model parameters $\boldsymbol{\theta}$ to the optimum $\boldsymbol{\theta}^*$ that minimizes some training objective function (e.g., mean squared loss between predicted labels and actual ones). DP-ML generally works by injecting noises (✱) at various stages of existing ML mechanisms, including the input dataset D (input perturbation `InPert`), training objective function (objective perturbation `ObjPert`), intermediate values used to update $\boldsymbol{\theta}$ such as gradients (update perturbation `UpPert`) and output model parameters $\boldsymbol{\theta}^*$ (output perturbation `OutPert`).

This report is organized as follows: in Sec. 2 we introduce necessary background knowledge related to ML and DP. Sec. 3 focuses on conventional ML models and how DP is achieved for each of them. Then, we illustrate differentially private deep learning and summarize two foundational works in Sec. 4. Finally, we conclude with a discussion about the implication of DP-ML in Sec. 5.

2 Backgrounds

2.1 Machine Learning

In this section, we present a formulation of ML problems and mechanisms in general. We do not cover specific ML models here; instead, they will be introduced later together with their DP versions. The notations introduced in this section will be used consistently in this report.

Let $D \in \mathcal{D}$ represent the training dataset consisting of n data. In ***supervised*** ML, dataset D takes the form $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ represents the ***features/attributes*** of each datum and $\mathbf{y} = \{y_1, ..., y_n\}$ represents the ***label*** to each datum (i.e., the i-th datum is $(\mathbf{x}_i, y_i)$). We also use $(\mathbf{x}, y)$ to represent any non-specific datum. Each $\mathbf{x}$ consists of m features $x_1, \cdots, x_m$. The model owner aims to represent the relationship between $\mathbf{x}$ and y through a model $\mathcal{F}$ with parameters $\boldsymbol{\theta}$, that is, $y = \mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x})$. To achieve this, they need to find the $\boldsymbol{\theta}$ that minimizes the difference between the model output $\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i)$ and actual label y_i for each datum $(\mathbf{x}_i, y_i)$, that is,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right), \tag{1}$$

where $\boldsymbol{\theta}^*$ is the trained model parameters and $\mathcal{L}$ is the *loss function* that we use to measure the difference (e.g., mean squared loss). The training mechanism $\mathcal{M} : \mathcal{D} \to \boldsymbol{\Theta}$ represents the algorithm that the model owner uses to find the optimal $\boldsymbol{\theta}^*$, i.e., $\boldsymbol{\theta}^* = \mathcal{M}(D)$.

Supervised ML can be categorized into ***classification*** and ***regression*** problems based on the nature of label y. In the case of classification, y takes value from a set of classes $\mathcal{C} = \{c_1, c_2, \cdots, c_{|\mathcal{C}|}\}$. For example, each datum from a dataset of animals is labelled as one of $\mathcal{C} = \{\textsc{Pig}, \textsc{Rabbit}, \textsc{Cat}\}$. In *binary classification* problems, each datum is either positive or negative, i.e., $\mathcal{C} = \{1, -1\}$ (e.g., whether a patient has a disease). On the other hand, regression problems deal with data where label y is continuous, i.e., $y \in \mathbb{R}$.

In practice, the dataset D unavoidably contains noises possibly in both features $\mathbf{x}$ and label y. As a result, a common yet undesirable problem in supervised ML is ***overfitting***, where the trained model $\mathcal{F}_{\boldsymbol{\theta}^*}$ is more than necessarily complex and therefore fits to every datum $(\mathbf{x}_i, y_i)$ in D perfectly, including the noises. Such models do not represent the actual relationship between $\mathbf{x}$ and y well and can make significant prediction errors when deployed. To address this issue, ***regularization*** techniques are often used to limit model complexity. Such techniques include an additional regularizer term $\mathcal{R}(\boldsymbol{\theta})$ in the objective function which penalizes the complexity of $\boldsymbol{\theta}$, that is,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right) + C \cdot \mathcal{R}(\boldsymbol{\theta}), \tag{2}$$

where the regularization parameter C controls the magnitude of penalty. A common regularizer is the L2 norm of model parameters, i.e., $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2$. This prevents any entry of $\|\boldsymbol{\theta}\|_2$ from getting too large. For example, consider the linear model $\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}$. If a particular entry of $\boldsymbol{\theta}$, say θ_k is too large, it means that a noise in the corresponding feature x_k would have a large impact, which is undesirable and can be mitigated through regularization.

Another type of ML is ***unsupervised*** learning, where data have no label. The model owner aims to understand the implicit structures of the data so as to extract useful information about the data distribution (e.g., clusters, effective dimensions) and even generate new data in future. Formally, dataset D takes the form $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$. The model with parameters $\boldsymbol{\theta}$ now represents some structure of the entire dataset D such that the objective function $\mathcal{F}_{\boldsymbol{\theta}}(D)$ represents the goodness of the structure (smaller $\mathcal{F}_{\boldsymbol{\theta}}(D)$ means better structure). Mathematically,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{F}_{\boldsymbol{\theta}}(D). \tag{3}$$

Deep Learning Deep learning (DL) is a sub-field of ML where the model structure is very complex, the number of model parameters is very large, and the number of data and their features is also large (e.g., images where each pixel is a feature). The fundamental ML model used for DL is the ***neural network*** (NN) model, which will be explained in detail in Sec. 4.1. Famous variants of NN include convolutional neural networks (CNNs), recurrent neural networks (RNNs), generative adversarial networks (GANs) and transformers. CNNs are intensively used for image data in computer vision; RNNs are commonly used for sequential data such as natural language data or time series data; GANs involve two NN models that compete against each other to generate more authentic data for image or 3D object generation; transformers capture the implicit relationships among certain features (e.g., context of speech where each word is a feature), which is the backbone of large language models such as GPT-4.

2.2 Differential Privacy

As introduced in Sec. 1, differential privacy (DP) assesses a randomized mechanism $\mathcal{M}$ (not limited to ML) by how indistinguishable $\mathcal{M}$'s outputs are when each datum is present or absent in the dataset. We call such pair of training datasets, with only one datum differing, *neighboring datasets*.

Formally, let $\mathcal{M} : \mathcal{D} \to \mathbf{\Theta}$ be the mechanism that takes in a dataset $D \in \mathcal{D}$ and produces randomized output $\boldsymbol{\theta} \in \mathbf{\Theta}$. In the context of ML, $\mathcal{M}$ is the training mechanism and $\boldsymbol{\theta}$ refers to the vector of trained model parameters. Consider a pair of neighboring datasets $D, D' \in \mathcal{D}$. If the output model parameters $\mathcal{M}(D)$ and $\mathcal{M}(D')$ approximately follow the same distribution, then attackers cannot tell whether the output is produced from D or D', and thus whether the datum by which D and D' differ even exists in the training dataset. Therefore, the definition of DP follows:

Definition 1. [ϵ-Differential Privacy] A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathbf{\Theta}$ satisfies *ϵ-differential privacy* (ϵ-DP) if for any pair of neighboring datasets $D, D' \in \mathcal{D}$ and for any $O \subseteq \mathbf{\Theta}$, it holds that

$$\Pr(\mathcal{M}(D) \in O) \leq e^{\epsilon} \cdot \Pr(\mathcal{M}(D') \in O). \tag{4}$$

The constant ϵ is called the *privacy level* of mechanism $\mathcal{M}$, or *privacy budget* of data owners, which represents how much privacy loss they can tolerate from mechanism $\mathcal{M}$. When a small ϵ is chosen, the difference between $\mathcal{M}(D)$ and $\mathcal{M}(D')$ is small and data owners enjoy better privacy (lower risk of privacy breach), yet probably at the cost of a less useful ML model.

Properties The simplicity of DP's definition (Eq. (4)) brings some useful properties to ease the use of DP mechanisms. Firstly, since $\exp(\epsilon_1 + \epsilon_2) = \exp(\epsilon_1) \cdot \exp(\epsilon_2)$, we can easily measure the privacy level of the combination of two DP mechanisms:

Proposition 1. [Composition] *When two independent DP mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$, which satisfy ϵ_1 and ϵ_2 respectively, are applied to the same dataset D to produce a combined output $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$, the combined mechanism satisfies $(\epsilon_1 + \epsilon_2)$-DP.*

Secondly, since the mechanism output $\boldsymbol{\theta}$ already preserves privacy, any postprocessing of $\boldsymbol{\theta}$ without using the original dataset D will also preserve privacy, that is,

Proposition 2. [Postprocessing-Robustness] *If $\mathcal{M} : \mathcal{D} \to \mathbf{\Theta}$ satisfies ϵ-DP, then for any postprocessing function f with domain $\mathbf{\Theta}$ the composed mechanism $f \circ \mathcal{M}$ satisfies ϵ-DP.*

In the context of ML privacy attacks, the postprocessing-robustness property ensures that any attack that the attackers perform on the trained ML model will not cause any further privacy loss beyond the desirable privacy budget.

Relaxation In some applications, ϵ-DP is considered to be too strong to even produce any useful ML model [13]. Therefore, a less strong version, called (ϵ, δ)-DP [16], is sometimes preferred over the original ϵ-DP:

Definition 2. [(ϵ, δ)-Differential Privacy] A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathbf{\Theta}$ satisfies *(ϵ, δ)-differential privacy* ((ϵ, δ)-DP) if for any pair of neighboring datasets $D, D' \in \mathcal{D}$ and for any $O \subseteq \mathbf{\Theta}$, it holds that

$$\Pr(\mathcal{M}(D) \in O) \leq e^{\epsilon} \cdot \Pr(\mathcal{M}(D') \in O) + \delta. \tag{5}$$

In other words, it tolerates a small probability δ (e.g., 1%) of privacy violation (i.e., the difference in mechanism outputs for neighboring datasets is larger than the factor ϵ).

Despite its practicality, analysis of the (ϵ, δ)-DP for composed mechanisms can sometimes be harder because the aforementioned composition property (Prop. 1) no longer holds due to the introduction of δ.

Noise Addition Mechanism The most common mechanism to achieve DP is to inject random artificial noise $\mathbf{e}$ to the mechanism output $\boldsymbol{\theta}$: if the noise $\mathbf{e}$ is extremely large, then the distribution of any perturbed output $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \mathbf{e}$ is indistinguishable from the noise distribution. Yet, one certainly does not prefer a mechanism whose output looks like noises.

How large should the injected noise be? Intuitively, it should be comparable to the difference in the outputs produced from neighboring datasets, preferably every pair of them. Therefore, we define the following:

Definition 3. [Sensitivity] For a randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathbf{\Theta}$, the L_p *sensitivity* of $\mathcal{M}$ is

$$\Delta_p = \max_{D, D'} \|\mathcal{M}(D) - \mathcal{M}(D')\|_p \tag{6}$$

for all neighboring datasets $D, D' \in \mathcal{D}$, where $\|\cdot\|_p$ refers to the L_p norm, i.e., its L_p distance from the origin[1].

Dwork [15] proves that adding a noise $\mathbf{e}$ proportional to $\exp(-\|\mathbf{e}\|_1 / a)$ gives (Δ_1 / a)-DP. Thus, one can achieve ϵ-DP by injecting Laplacian noise $\mathbf{e}$ of scale (Δ_1 / ϵ) (and mean $\mathbf{0}$) (i.e., the ***Laplacian mechanism***):

$$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^* + \mathrm{Lap}\left(\frac{\Delta_1}{\epsilon}\right), \tag{7}$$

where $\mathrm{Lap}(a)$ has probability density function $p(\mathbf{e}) = (1/2a) \exp(-\|\mathbf{e}\|_1 / a)$.

In the case of (ϵ, δ)-DP, Dwork and Roth [17] show that for any $\epsilon \in (0, 1)$ and $\delta \in (0, 1)$, one can achieve (ϵ, δ)-DP by injecting Gaussian noise to the mechanism output (i.e., the ***Gaussian mechanism***):

$$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^* + \mathcal{N}\left(0, \frac{2\ln(1.25/\delta)\Delta_2^2}{\epsilon^2}\right). \tag{8}$$

For simplicity we will use Δ to represent sensitivity in general and it defaults to Δ_1 for Laplacian mechanism and Δ_2 for Gaussian mechanism.

[1]For example, the L2 norm of a r-sized vector $\|\boldsymbol{\theta}\|_2 = \sqrt{\theta_1^2 + \theta_2^2 + \cdots + \theta_r^2}$.

3 DP Mechanisms for Machine Learning

In this section, we introduce DP-ML mechanisms for some traditional ML algorithms. In particular, we will discuss **supervised learning models** including linear models (Sec. 3.1), naïve Bayes models (Sec. 3.2) and decision trees (Sec. 3.3) and **unsupervised learning models** including K-means clustering (Sec. 3.4) and principal component analysis (Sec. 3.5). We give a brief explanation about each model, focusing on **what objective it aims to optimize** and **what the model parameters $\boldsymbol{\theta}$ are**, before proceeding to the DP mechanisms.

Almost all DP-ML mechanisms are based on the **noise addition mechanism** introduced in Sec. 2.2 to achieve DP. Therefore, we need to answer a few questions for each ML model: *Where should we inject noise? What form of noise should we add? What is the sensitivity of the training mechanism (so that we know how much noise to add)?*

3.1 Linear Models

A ***linear model*** with parameters $\boldsymbol{\theta} = [\boldsymbol{\theta}_{\mathbf{x}} \quad \theta_0]$ aims to associate label y with an affine (i.e., linear with offset) transformation of features $\mathbf{x}$: $\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0 \in \mathbb{R}$. In binary classification problems with $\mathbf{y} = \{y_1, y_2, \cdots, y_n\} \in \{1, -1\}^n$, a ***linear classification*** (LinC) model takes the form

$$\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \operatorname{sign}\left(\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0\right), \tag{9}$$

where $\operatorname{sign}(a) = a/|a|$ outputs the sign of the input. The hyperplane $\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0 = 0$ serves as the *decision boundary* separating the positive and negative data. LinC may use *Hinge loss* as its loss function, that is,

$$\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}), y\right) = \max\left(0, 1 - y\left(\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0\right)\right). \tag{10}$$

Logistic regression (LogR) is another linear model used for classification problems. LogR aims to output the **likelihood** that the label to a datum $\mathbf{x}$ is $y = 1$:

$$\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \Pr(y = 1 \mid \mathbf{x}) = \sigma\left(\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0\right), \tag{11}$$

where $\sigma(a) = 1/(1+\exp(-a))$ is the *sigmoid* function that maps the affine transformation $\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0 \in \mathbb{R}$ to a likelihood in $[0, 1]$. LogR uses *logistic loss* as its loss function, that is,

$$\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}), y\right) = \log\left(1 + \exp\left(-y\left(\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0\right)\right)\right). \tag{12}$$

Linear regression (LinR) is used for regression problems. It directly takes the form

$$\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0, \tag{13}$$

which associates y with $\mathbf{x}$ through an affine relationship. LinR commonly uses *mean squared loss* (L2 loss) as its loss function, i.e.,

$$\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}), y\right) = \left(y - \left(\boldsymbol{\theta}_{\mathbf{x}}^\top \mathbf{x} + \theta_0\right)\right)^2. \tag{14}$$

In all the aforementioned linear models, **regularization** (see Sec. 2.1) is typically done by penalizing the L2 norm of model parameters $\boldsymbol{\theta}$ to avoid overfitting. Therefore, a general form of linear models' objective function can be written as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right) + C \cdot \|\boldsymbol{\theta}\|_2^2. \tag{15}$$

Differentially Private Linear Models Lei [26] proposes a simple `InPert` DP mechanism that applies to LinR and LogR. It works by creating a synthetic dataset based on the perturbed histogram of the original dataset. However, it only works for low-dimensional problems and may lead to poor accuracy on large-scale datasets [22]. Interested readers may refer to App. B.3 to learn more about this method.

Chauduri *et al.* [8, 9] propose one `OutPert` and one `ObjPert` DP mechanisms for **regularized** linear models with normalized features (i.e., $\|\mathbf{x}\|_2 = 1$), with the assumption that both the loss function $\mathcal{L}$ and the regularizer are strongly convex and differentiable (e.g., logistic loss and L2 regularizer in Eq. (15)). The sensitivity of trained model parameters $\boldsymbol{\theta}$ is $\Delta = 2/C$ roughly because the impact of the 2 data that differ in every pair of neighboring sets is restricted by the regularization parameter C. Therefore, ϵ-DP through `OutPert` can be achieved by setting

$$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^* + \operatorname{Lap}\left(\frac{2}{C\epsilon}\right) \tag{16}$$

from Eq. (7). As a different approach, one can also perturb the minimization objective in Eq. (15), i.e.,

$$\tilde{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right) + C \cdot \|\boldsymbol{\theta}\|_2^2 + \mathbf{e}^\top \boldsymbol{\theta}, \tag{17}$$

where $\mathbf{e} \sim \operatorname{Lap}(2/C\epsilon)$. In other words, the derivative of the objective function (and thus the minimizer which makes the derivative zero) is perturbed by a Laplacian noise of the same scale as Eq. (16). The above `ObjPert` mechanism additionally requires the loss function $\mathcal{L}$ and regularizer to be doubly differentiable. In practice, in order to use the above `OutPert` and `ObjPert` mechanisms with non-differentiable but continuous loss functions or regularizers (e.g., Hinge loss is not differentiable at 1), one can convert it to a differentiable one by "smoothening" the non-differentiable points using piecewise functions.

Zhang *et al.* [40] propose a more general `ObjPert` mechanism that can be used in both **standard** LogR and LinR models **without regularization** and regularized ones, called the ***functional mechanism***. In such a general setting, the sensitivity of the original objective function is hard to measure. Therefore, one may rewrite the original logistic loss function (12) with model parameters $\boldsymbol{\theta} = (\theta_1, ..., \theta_r)$ as its unique Stone-Weierstrass approximation [21][2]:

$$\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}), y\right) = \sum_{\substack{\phi(\boldsymbol{\theta}) = \theta_1^{\zeta_1} \theta_2^{\zeta_2} \cdots \theta_r^{\zeta_r} \\ \zeta_1 + \cdots + \zeta_r \leq Z}} \lambda_{\mathbf{x}, y, \phi} \phi(\boldsymbol{\theta}), \tag{18}$$

which is a weighted sum of polynomials $\phi(\boldsymbol{\theta})$ of order up to Z ($\lambda_{\mathbf{x},y,\phi}$ is the weight; Z can be potentially infinite). The sum of losses for all data in Eq. (1) that we want to minimize is therefore also a weighted sum of these $\phi(\boldsymbol{\theta})$, with the weight of $\phi(\boldsymbol{\theta})$ being $\lambda_\phi = \sum_{i=1}^n \lambda_{\mathbf{x}_i, y_i, \phi}$. To achieve

[2]Readers who are unfamiliar with approximation theory may draw a parallel to power series expansion of a function. The expansion serves as an approximation to the function.

ϵ-DP through ObjPert, one can then inject noises to these weights, i.e., for each polynomial $\phi(\boldsymbol{\theta})$,

$$\tilde{\lambda}_\phi \leftarrow \lambda_\phi + \mathrm{Lap}\left(\frac{\Delta}{\epsilon}\right), \tag{19}$$

where $\Delta = 2\max_{\mathbf{x},y}\sum_\phi |\lambda_{\mathbf{x},y,\phi}|$ is the sensitivity. Then the objective in Eq. (1) becomes

$$\tilde{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{\substack{\phi(\boldsymbol{\theta})=\theta_1^{\zeta_1}\theta_2^{\zeta_2}\cdots\theta_r^{\zeta_r}\\ \zeta_1+\cdots+\zeta_r\le Z}} \tilde{\lambda}_\phi \phi(\boldsymbol{\theta}). \tag{20}$$

In practice, the highest order of $\phi_{\boldsymbol{\theta}}$, Z, can be forced to be finite by taking the Taylor expansion of loss function $\mathcal{L}$ and truncating it. This makes the above method computationally feasible.

When the dataset D has larger number of data n or number of features m, it can be computationally expensive to analytically find the minimizer $\boldsymbol{\theta}^*$ that minimizes the objective function. In such cases, iterative mechanisms such as gradient descent are often used. We will discuss such mechanisms and their DP versions in Sec. 4 DP Mechanisms for Deep Learning instead when we illustrate perceptrons and neural networks.

3.2 Naïve Bayes Model

From a probabilistic point of view, both features $\mathbf{x} = (x_1, x_2, \cdots, x_m)$ and label y are observations/realizations of some random variables $X = (X_1, X_2, \cdots, X_m)$ and Y respectively. Y may take values from the set of classes $\mathcal{C} = \{c_1, c_2, \cdots, c_{|\mathcal{C}|}\}$. The label Y to a given set of features $\mathbf{x}$ follows some unknown probability distribution $p(Y \mid X = \mathbf{x})$ and the model aims to predict the label y as class $c \in \mathcal{C}$ that is the most probable given features $\mathbf{x}$. This is known as the *maximum a posterior* estimate:

$$\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \arg\max_{c\in\mathcal{C}} \Pr(Y = c \mid X = \mathbf{x}). \tag{21}$$

However, the distribution $p(Y \mid X = \mathbf{x})$ could be difficult to estimate from the dataset D directly due to lack of data with features $\mathbf{x}$. On the other hand, the distribution of label Y regardless of the features, $p(Y)$, can be easily estimated from the label set $\mathbf{y} = \{y_1, \cdots, y_n\}$. For example, for classification problems, we assume $p(Y)$ to be a categorical distribution and use the frequency of class c, $\frac{n_c}{n}$, as the probability of class c. Here n_c refers to the number of data labelled as class c.

For simplicity we will abbreviate $X = \mathbf{x}$ as $\mathbf{x}$ and $Y = c$ as c in probability notations from this point onwards. The Bayes' theorem [6] provides a way to update from the *prior* distribution, $p(Y)$, to the *posterior* distribution after observing $\mathbf{x}$, $p(Y \mid \mathbf{x})$. Mathematically,

$$\Pr(c \mid \mathbf{x}) = \frac{\Pr(\mathbf{x} \mid c)\Pr(c)}{\Pr(\mathbf{x})}. \tag{22}$$

Combining Eq. (21) and (22), the denominator $\Pr(\mathbf{x})$ becomes unnecessary as a constant, thus,

$$\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \arg\max_{c\in\mathcal{C}} \Pr(\mathbf{x} \mid c)\Pr(c). \tag{23}$$

The ***naïve Bayes*** (NB) model further assumes that all features $X_1, X_2, \cdots, X_m$ are conditionally independent given the label $y = c$. This enables us to further break down $\Pr(\mathbf{x} \mid c)$ in Eq. (23):

$$\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}) = \arg\max_{c\in\mathcal{C}} \Pr(c)\prod_{j=1}^{m}\Pr(x_j \mid c), \tag{24}$$

where the distribution $p(X_j \mid c)$ can also be easily estimated by looking at the features set $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ restricted to feature X_j and class c. In the same fashion as how we estimate $p(Y)$, when X_j is discrete and finite, we assume $p(X_j \mid c)$ as a categorical distribution with $\Pr(x_j \mid c)$ being the frequency of x_j in class c, $\frac{n_{c,x_j}}{n_c}$. Here n_{c,x_j} refers to the number of data whose label is $y = c$ and j-th feature equals x_j. When X_j is continuous or infinite, we assume $p(X_j \mid c)$ as a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2 estimated using the sample mean and variance.

Specifically, the model parameters $\boldsymbol{\theta}$ in NB refers to the parameters of distributions $p(Y)$ and $p(X_j \mid c)$ for each feature X_j and class c. For example, it can contain the probability of each category for a categorical distribution and the mean and variance for a Gaussian distribution.

Differentially Private Naïve Bayes For NB models, ϵ-privacy through ObjPert is achieved by injecting noises to the parameters of distributions [37]. Consider first the categorical features. In any pair of neighboring datasets, for any label y or feature value x_j, the number of data with label $y = c$, n_c, or the number of data whose labels are $y = c$ and j-th features equal x_j, n_{c,x_j}, clearly differs by at most 1 because only 1 datum changes. Therefore, we can simply do the following updates:

$$\tilde{n}_c \leftarrow n_c + \mathrm{Lap}\left(\frac{1}{\epsilon}\right); \tag{25}$$

$$\tilde{n}_{c,x_j} \leftarrow n_{c,x_j} + \mathrm{Lap}\left(\frac{1}{\epsilon}\right). \tag{26}$$

The perturbed parameters of distributions (i.e., frequencies) can then be calculated by $\frac{\tilde{n}_c}{n}$ and $\frac{\tilde{n}_{c,x_j}}{\tilde{n}_c}$.

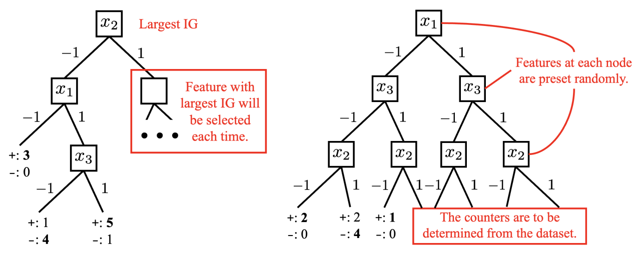
Consider now a continuous or infinite feature X_j that is assumed to follow a Gaussian prior. Assume that all possible observed values for X_j are bounded between $[l_j, u_j]$. Considering neighboring datasets of size n and $n+1$, the sensitivity of its mean μ_j is $\Delta_{\mu_j} = \frac{u_j - l_j}{n+1}$; the sensitivity of its standard deviation σ_j is $\Delta_{\sigma_j} = \sqrt{\frac{n(u_j-l_j)}{n+1}}$. Thus the perturbed mean and standard deviation are

$$\tilde{\mu}_j \leftarrow \mu_j + \mathrm{Lap}\left(\frac{\Delta_{\mu_j}}{\epsilon}\right); \tag{27}$$

$$\tilde{\sigma}_j \leftarrow \sigma_j + \mathrm{Lap}\left(\frac{\Delta_{\sigma_j}}{\epsilon}\right). \tag{28}$$

3.3 Decision Tree Models

A ***decision tree*** (DT) model works primarily for classification problems with discrete (or discretized continuous) features. As its name suggests, a DT model involves a tree



Largest IG
Feature with largest IG will be selected each time.
+: 3 -: 0
+: 1 -: 4
+: 5 -: 1
Features at each node are preset randomly.
+: 2 -: 0
+: 2 -: 4
+: 1 -: 0
The counters are to be determined from the dataset.

(a) IDT training.   (b) RDT training.

Figure 2: **Illustration of two types of DT models.** In example 2a, when classifying a new datum with binary features $\mathbf{x} = \{x_1 = 1, x_2 = -1, x_3 = -1\}$, the DT model starts from the root node x_2, followed by x_1 and x_3 and takes the majority label at the leaf node, i.e., $\mathcal{F}(\mathbf{x}) = -1$.

In terms of model training, IDT (2a) builds the tree iteratively by selecting features according to some heuristic, whereas RDT (2b) presets the tree structure and simply updates the counters $\boldsymbol{\ell}_\nu$ at each leaf node ν.

structure where each *node* represents one feature based on whose discrete values we can split the dataset (i.e., a decision to be made), and each *branch* denotes a value of the feature on its parent node.

A common approach to train a DT model is the *iterative* approach that builds an ***iterative decision tree*** (IDT) (Fig. 2a). Starting from the root node and the entire training dataset D, in each iteration, we select one of the m features as the current node. Each branch (i.e., value of the selected feature) will only point to a subset of data and a new child node. We then repeat this process on the subset of data $D_k \subseteq D$ and the child node, until (i) all the data in the subset have the same label, or (ii) there is no feature we can select as the current node. We call such nodes *leaf nodes* (i.e., those without child nodes). When IDT is applied to classify a new datum, the datum will be directed to a unique leaf node and assigned the following label: in case (i), we just take the common label; in case (ii), we choose the label possessed by the most data in that subset (i.e., a majority vote). Just like any other tree structure, DT's space complexity increases exponentially with its height h (i.e., maximum number of features required to reach (i) or (ii)). Also, a smaller and thus simpler tree is often preferred to avoid overfitting. Thus, features are carefully selected at each node to reach (i) as fast as possible. One well-known heuristic is to select the feature that brings the largest *information gain*, that is, reduces our *uncertainty* about the label to the largest extent (e.g., in case (i), we are 100% certain about the label of any datum that falls to that leaf node). Mathematically, uncertainty about the label of a dataset D is quantified by *entropy*:

$$\text{Entropy}(D) = -\sum_{c\in\mathcal{C}} \Pr(c)\log\Pr(c), \tag{29}$$

where (recall that) $\Pr(c) = \Pr(Y = c) = n_c/n$ is the proportion of data with label c. Suppose that selecting the j-th feature segments the dataset D to Q disjoint datasets $\left\{D^{(1)}, D^{(2)}, \cdots, D^{(Q)}\right\}$, then information gain is defined as the reduction in entropy after selecting the j-th feature:

$$\text{IG}(x_j) = \text{Entropy}(D) - \sum_{q=1}^{Q} \frac{|D^{(q)}|}{|D|}\text{Entropy}\left(D^{(q)}\right). \tag{30}$$

App. B.1 gives an example of how to construct an IDT model using information gain as the heuristic of selection.

Another approach to train a DT model is the *random* approach that builds a ***random decision tree*** (RDT) (Fig. 2b). Specifically, RDT pre-selects $h \leq m$ features (with order) randomly. Clearly, these features will give a RDT of height h. Let $\mathcal{V}$ denote the set of leaf nodes in the RDT. Each leaf node $\nu \in \mathcal{V}$ contains a vector $\boldsymbol{\ell}_\nu$ of length equal to the number of classes $|\mathcal{C}|$. The vector $\boldsymbol{\ell}_\nu$ represents the number of data in leaf node ν in each class, so $\ell_{\nu,c} = 0$ for every leaf node ν and every class c before training. During the training process, each datum with label $y = c$ will be directed to a unique leaf node ν by the selected h features and increase the entry $\ell_{\nu,c}$ by 1. Therefore, unlike IDT which does not have model parameters $\boldsymbol{\theta}$ of fixed size, RDT has $\boldsymbol{\theta}$ of size $|\mathcal{V}|\cdot|\mathcal{C}|$ which uniquely characterizes the trained model. When applied to classify a new datum, RDT works similarly to IDT by choosing the most common label c at the leaf nodes.

Clearly, RDT is sub-optimal compared with IDT in terms of accuracy and size. However, it involves less computation and thus a lot of RDT models can be combined to form an ensemble model, which has proven to give a good performance [19]. Moreover, since RDT has model parameters $\boldsymbol{\theta}$ of fixed size, it can be perturbed more easily to achieve DP.

Differentially Private Decision Tree To achieve ϵ-DP for IDT, Blum *et al.* [7] propose to perturb the intermediate step (`UpPert`) at each iteration (e.g., information gain). Since the information gain needs to be repeatedly calculated for every remaining feature at each iteration which consumes the privacy budget ϵ quickly, a large noise needs to be injected to each of these information gains to ensure a small privacy loss each time, so that the total privacy after composition (see Sec. 2.2) does not exceed ϵ. Moreover, the IDT must not grow too high until any datum can be uniquely identified at any leaf node. In consequence, the shallow and noisy IDT model can easily fail.

As a result, Jagannathan *et al.* [23] propose to achieve ϵ-DP through `OutPert` for RDT instead. The sensitivity of each model parameter $\ell_{\nu,c}$ is 1 because the one different datum can only change a leaf node's counter by either 0 or 1. Hence, ϵ-privacy through `OutPert` can be achieved by injecting a $\text{Lap}(1/\epsilon)$ noise to each parameter $\ell_{\nu,c}$ of the trained RDT model:

$$\tilde{\ell}_{\nu,c} \leftarrow \ell_{\nu,c} + \text{Lap}\left(\frac{1}{\epsilon}\right), \forall \nu \in \mathcal{V}, c \in \mathcal{C}. \tag{31}$$

Consider an ensemble of N RDT models that are trained with the above `OutPert` method. From the composition property (Prop. 1), the ensemble model satisfies $(N\epsilon)$-DP. Conversely, if the data owners desire ϵ-DP, each parameter $\ell_{\nu,c}$ of each RDT model should be perturbed with a $\text{Lap}(N/\epsilon)$ noise. In fact, such idea can be applied to any other ensemble models to achieve DP as well.

3.4 K-Means Clustering

K-means clustering (KMC) works for unsupervised learning problem where data labels are not available. The model owner wishes to understand the distribution of dataset D, specifically whether some data are closer to one another (i.e., they are *clustered*) and where the cluster centers are. For example, a city planner may place bus terminals at each cluster center such that all citizens may use a bus terminal near them.

Mathematically, KMC aims to partition a dataset D of n data into K clusters such that within each cluster, the sum of variances from all data within that cluster to the cluster *centroid* (i.e., center) is minimized. Let the set of K clusters be $\mathbf{C} = \{C_1, C_2 ..., C_K\}$ with centroids $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K$ respectively. The objective is to find the optimum $\mathbf{C}^*$ such that

$$\begin{aligned}\mathbf{C}^* &= \arg\min_{\mathbf{C}} \sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} \|\mathbf{x} - \mathbf{c}_i\|_2^2 \\ &= \arg\min_{\mathbf{C}} \sum_{k=1}^{K} \left(|C_k| \cdot \mathrm{Var}(C_k)\right),\end{aligned} \tag{32}$$

where each centroid $\mathbf{c}_k$ is the mean of all points in cluster C_k, $\left(\sum_{\mathbf{x}\in C_k} \mathbf{x}\right) / |C_k|$.

To train a KMC clustering model, the K centroids are initially chosen randomly. Each training datum is assigned to the cluster of its nearest centroid. Then, each cluster's centroid is updated based on the mean of data in each cluster. This process is repeated until all centroids do not change anymore.

It is clear that a trained KMC model can be uniquely determined by its K cluster centroids $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K$. Therefore, its model parameters $\boldsymbol{\theta}$ are precisely a combination of these centroids $[\mathbf{c}_1 \quad \mathbf{c}_2 \quad \cdots \quad \mathbf{c}_K]$.

Differentially Private K-Means Clustering The sensitivity of KMC model parameters $\boldsymbol{\theta} = [\mathbf{c}_1 \quad \mathbf{c}_2 \quad \cdots \quad \mathbf{c}_K]$ is hard to quantify, because changing one datum might change the whole dataset's pattern and lead some centroids to move far (e.g., consider two neighboring datasets that differ by a very far anomalous datum). Even if we are able to bound such a large sensitivity, the trained model will be destroyed by the corresponding noise added. Therefore, converting plain KMC model to a DP version is not preferred.

To reduce the impact of anomalous data and thus sensitivity, [29] proposes a sample-aggregate framework which can consequently achieve ϵ-DP by `OutPert`. At the sampling stage, the training data is randomly partitioned into q small subsets $D_1, D_2, ..., D_q$. When q is set to be sufficiently large, it is nearly certain that only a small proportion of these subsets contain anomalous data. If we independently train KMC models using every subset to obtain $\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_q$ (each containing K centroids), then it is likely that most of these centroid sets are close to one another. Following this idea, one can simply select the centroid sets within a "ball" of L1 diameter Δ and omit the other centroid sets that are likely to be off. The aggregation stage will simply output the center of the ball, $\mathbf{C}^*$. In this way, the sensitivity of the corresponding vector $\boldsymbol{\theta}^*$ is at most Δ.

One can then follow the normal step to achieve ϵ-DP:

$$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^* + \mathrm{Lap}\left(\frac{\Delta}{\epsilon}\right). \tag{33}$$

3.5 Principal Component Analysis

Principal component analysis (PCA) is an unsupervised learning model commonly used to reduce high-dimensional data to a new coordinate system of lower dimensions structured by the orthogonal vectors called *principal components* (PCs). The first PC is the direction along which the data have the highest variance, thus retaining the highest information from the original data. All subsequent PCs capture the next highest variance within the data and must be orthogonal to all previous PCs to ensure that all PCs are not correlated.

Mathematically, PCA starts from the $m \times m$ feature covariance matrix $\mathbf{\Sigma} = (1/n)\mathbf{X}^\top\mathbf{X}$ of the dataset. One can then compute $\mathbf{\Sigma}$'s eigenvectors (PCs) and the corresponding eigenvalues which reflects the amount of variance in each PC. Ranking all the eigenvectors according to their eigenvalues yields the PCs in the order of significance.

In PCA models, the model parameters $\boldsymbol{\theta}$ contain each PC in order.

Differentially Private Principal Component Analysis Blum *et al.* propose the SuLQ framework [7] which achieves DP through `UpPert`. SuLQ perturbs every entry in the covariance matrix $\mathbf{\Sigma}$ to obtain $\tilde{\mathbf{\Sigma}}$. However, the perturbed matrix $\tilde{\mathbf{\Sigma}}$ may not be symmetric, which means that the eigenvectors and corresponding eigenvectors may be complex (i.e., not real). Such results are useless for PCA.

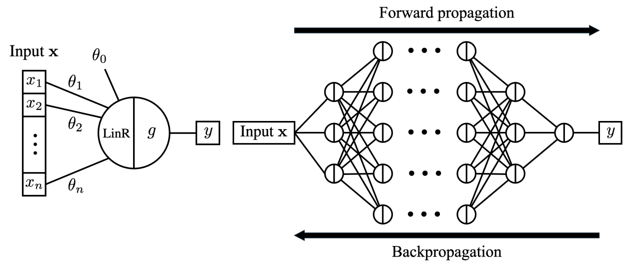
To address this, [10] proposes the PPCA mechanism to achieve ϵ-DP, making use of the exponential mechanism [28] of DP. The exponential mechanism is a generalized version of the Laplacian mechanism, which can directly modify mechanism $\mathcal{M}$ to produce a differentially private output $\tilde{\mathcal{M}}(D)$, instead of injecting noises to its output[3]. For example, in PPCA, the PCs can be directly sampled from a perturbed distribution in the form of the matrix Bingham distribution $\mathrm{Bmf}(n\epsilon\mathbf{\Sigma}/2)$ [11].

4 Mechanisms for Deep Learning

Recall from Sec. 2.1 that the ***neural network*** (NN) model is the core of DL. In this section, we will first explain the structure and training of NN models (Sec. 4.1). We focus on the most fundamental and important type: ***perceptron neural networks*** (PNNs). In fact, all current NN models are variants of PNNs.

We then proceed to explain two major, if not the only two, DP mechanisms to train NN models privately, namely ***differentially private stochastic gradient descent*** (Sec. 4.2) and ***private aggregation of teacher ensembles*** (Sec. 4.3).

[3]We will not give a detailed explanation of the exponential mechanism for the purpose of this report since it requires much additional knowledge on DP and is rarely used in existing DP-ML literatures. Interested readers may refer to [24] which gives a good explanation of the exponential mechanism.



(a) A perceptron. (b) A (P)NN model.

Figure 3: **Illustration of PNNs**. A PNN model (3b) comprises of layers of perceptrons (3a).

4.1 Neural Network Model

Perceptrons ***Perceptrons*** are the basic building blocks of PNNs, similar to neurons in human brains. As shown in Fig. 3a, a perceptron with parameters $\boldsymbol{\theta} = [\boldsymbol{\theta}_{\mathbf{x}} \quad \theta_0]$ and input $\mathbf{x}$ is essentially a LinR model with an additional activation function g (which can be **non-linear**):

$$\mathcal{F}_{\boldsymbol{\theta}} = g\left(\boldsymbol{\theta}_{\mathbf{x}}^{\top}\mathbf{x} + \theta_0\right). \tag{34}$$

Examples of common activation functions g include ReLU ($g(a) = \max(0, a)$), sigmoid ($g(a) = 1/(1 + \exp(-a))$) and Swish ($g(a) = a/(1 + \exp(-a))$). The activation functions provide non-linearity and therefore (theoretically) allow PNNs (i.e., composition of many perceptrons) to represent any unknown relationship between features $\mathbf{x}$ and label y.

Perceptrons use similar loss functions $\mathcal{L}$ (e.g., mean squared loss) to LinR models. Training a perceptron is to minimize the sum of losses over all data $(\mathbf{x}_i, y_i) \in D$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right). \tag{35}$$

The sum of losses in Eq. (35) is called the *objective function* and often assumed to be convex and differentiable. Therefore, its gradient at $\boldsymbol{\theta}^*$ should equal zero, that is,

$$\sum_{i=1}^{n} \nabla\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}^*}(\mathbf{x}_i), y_i\right) = \mathbf{0}, \tag{36}$$

where ∇ is the vector differentiation operator.

Gradient Descent Solving Eq. (36) analytically requires matrix inversion, which is computationally infeasible when the dataset D is large. Therefore, one has to resort to iterative optimization techniques instead, where $\boldsymbol{\theta}$ is updated at each step until converging to $\boldsymbol{\theta}^*$.

Gradient descent (GD) is a commonly used iterative optimization technique. It works by updating $\boldsymbol{\theta}$ in the opposite direction to the gradient at $\boldsymbol{\theta}$. For example, consider the simple convex function $f(\theta) = \theta^2$ with minimizer $\theta^* = 0$. When θ is positive, the gradient 2θ is positive and we need to decrease θ, vice versa.

Algorithmically, GD starts with an initial $\boldsymbol{\theta}$ ($\mathbf{0}$ or random) and a preset *learning rate* η which controls how far we should update $\boldsymbol{\theta}$ each time. At each iteration, GD computes the average[4] gradient of the objective function over all data $(\mathbf{x}_i, y_i) \in D$ to update the parameters:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \frac{1}{n}\sum_{i=1}^{n} \nabla\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right). \tag{37}$$

Sometimes one may prefer a learning rate that decays over the number of iterations as we are getting closer to the minimum.

However, computing gradients using all data becomes expensive when the dataset D is large. Moreover, when the objective function is not necessarily convex, GD may eventually stop at a local minimum [5]. Thus a variant of GD called ***stochastic GD*** (SGD) works differently by randomly sampling **one** datum $(\mathbf{x}_i, y_i) \in D$ and calculating the gradient for only this datum in each iteration:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla\mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i\right). \tag{38}$$

Despite its stochasticity, SGD has been proven to converge for most of the time. Therefore, when the dataset D is large, SGD is preferred because of its time efficiency.

Mini-batch GD is another variant of GD which combines the advantages of GD and SGD: In each iteration, a random subset (mini-batch) of size b is sampled instead of a single datum. The gradient is computed by averaging the gradients for only the data in the mini-batch.

Neural Networks and Backpropagation As shown in Fig. 3b, a PNN is made up of several **layers** of perceptrons. Outputs of perceptrons in the previous layer are inputs of perceptrons in the next layer. Through such composition of non-linear functions, PNNs are capable of representing very complex relationships between features $\mathbf{x}$ and label y.

PNNs are trained through ***backpropagation*** [25]. In each iteration, the input $\mathbf{x}$ first passes through all the layers to get the output $\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x})$ and loss $\mathcal{L}(\mathcal{F}_{\boldsymbol{\theta}}(\mathbf{x}), y)$. This is called a *forward propagation*. Computing the gradients for the last layer of perceptrons is as easy as doing so for a single perceptron model. The gradients for the second last layer of perceptrons can be computed based on the last layer through the chain rule (see App. B.2 for more detailed explanation and examples). By repeating this process until the gradients for the first layer (and therefore all layers) of perceptrons are calculated, one can then update all model parameters $\boldsymbol{\theta}$ using GD. Since the gradients are computed backwards layer by layer, this technique is called *backpropagation*.

4.2 Differentially Private Stochastic Gradient Descent

Unlike traditional ML models listed in Sec. 3 whose parameters $\boldsymbol{\theta}$'s sensitivity is easy to measure, NN models are too complex to break down to quantify their sensitivity directly.

[4]Note that the **sum** of gradients can also be used instead of the average. Averaging can avoid the impact of number of data, especially when a mini-batch of data instead the entire dataset is used to compute the gradient.

[5]Local minima are points which are not minimizers but have zero gradient

Algorithm 1: **DP-SGD.**

```
Input: Noise scale $\sigma$, sample size $b$, gradient norm bound $G$, number of iterations $T$, learning rate $\boldsymbol{\eta} = (\eta_1, \eta_2, \cdots, \eta_T)$.
Output: Trained model parameters $\boldsymbol{\theta}^{(t)}$ after $T$ iterations.
1: Initialize $\boldsymbol{\theta}^{(0)}$ randomly
2: for $t = 0, 1, 2, \cdots, T-1$ do
3:     Take a random sample $D_t$ of size $b$ with sampling probability $b/n$
4:     for each $\{\mathbf{x}, y\} \in D_t$ do
5:         Gradient $g_t(\mathbf{x}) \leftarrow \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{L}\left(\mathcal{F}_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}), y\right)$
6:     end for
7:     Clipped gradient $\bar{g}_t(\mathbf{x}) = g_t(\mathbf{x}) / \max\left(1, \frac{\|g_t(\mathbf{x})\|_2}{G}\right)$
8:     Noisy gradient $\tilde{g}_t = \frac{1}{b}\left(\sum_{\mathbf{x} \in D_t} \bar{g}_t(\mathbf{x}) + \mathcal{N}(0, \sigma^2 G^2 \boldsymbol{I})\right)$
9:     New parameters $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \tilde{g}_t$
10: end for
11: return $\boldsymbol{\theta}^{(T)}$
```

An alternative methodology is to treat NN as a black box and analyze its input-output behavior. However, NN models usually demonstrate intricate behavior again due to their complex structures. Therefore, it is hardly possible to achieve DP through `OutPert`, unless one injects an unreasonably large noise which results in a useless model. Meanwhile, `InPert` and `ObjPert` are also regarded as impractical [38].

Therefore, one may resort to `UpPert` by practising DP in each iteration of GD or SGD. Precisely, [5] develops `UpPert` for GD and [1] develops `UpPert` for SGD. A special advantage of practising `UpPert` for SGD over GD is because of its stochasticity: since random data are chosen in each iteration, the privacy of each datum is amplified because each datum has a probability of never being used [4]. For this reason we will focus on explaining the ***differentially private SGD*** (DP-SGD) algorithm in this section. It is also worth noting that (ϵ, δ)-DP is preferred over ϵ-DP since the former provides more relaxed constraint and thus less sacrifice in model performance. This is especially meaningful to iterative training mechanisms including SGD since the noises will accumulate across iterations.

A complete pseudocode for the DP-SGD mechanism is given in Alg. 1. In each iteration, we select a random sample $D_t \subseteq D$ of size b to calculate and average the gradients (i.e., a mini-batch GD). To ensure each datum's privacy, we do not want any particular datum to have too much impact on the average gradient, thus we "clip" (i.e., scale) each gradient in line 7 such that each gradient's norm does not exceed a preset constant G. We then take the average gradient and add a Gaussian noise based on the Gaussian mechanism introduced in Sec. 2.2 in line 8. Finally, we apply the descent in line 9.

Proposition 3. [Privacy of DP-SGD [1]] *There exists constants C_1 and C_2 such that for any $\epsilon < C_1(b/n)^2 T$, DP-SGD satisfies (ϵ, δ)-DP for any $\delta > 0$ if we set the noise scale $\sigma \le C_2 b\sqrt{T \log(1/\delta)}/(n\epsilon)$.*

The above Prop. 3 is proven by a novel *moment account* method, which provides a bound for the privacy loss in each iteration. Roughly, after we set the noise scale σ, each SGD iteration will cost some bounded amount of privacy such that after a computable number of iterations, our privacy budget ϵ will be exhausted. In particular, the moment account method provides a better privacy bound than the vanilla composition theorem for (ϵ, δ)-DP.

Meanwhile, the *utility* (i.e., model performance) of NN models trained through DP-SGD can also be ensured. In fact, it can be proven in a similar fashion to how the utility of plain SGD is guaranteed, where the latter has been thoroughly analyzed (e.g., [32]). Intuitively, assume the objective function is convex. If we perform gradient descent using the average gradient of all data (i.e., GD), then it is guaranteed that $\boldsymbol{\theta}$ will converge to the minimizer $\boldsymbol{\theta}^*$. In the case of SGD, the **expectation** of the gradient of the randomly sampled data is equal to the average gradient in GD, thus in the expected sense $\boldsymbol{\theta}$ will still converge to the minimizer $\boldsymbol{\theta}^*$. In the case of DP-SGD, the **expectation** of the **perturbed** gradient of the randomly sampled data is still equal to the average gradient in GD (though with a larger variance), so it is expected that $\boldsymbol{\theta}$ will still converge to the minimizer $\boldsymbol{\theta}^*$.

Due to the simplicity, privacy and utility guarantee of DP-SGD, it has become the most prevalent DP mechanism for DL. It has also been implemented on trending DL libraries such as Tensorflow Privacy.

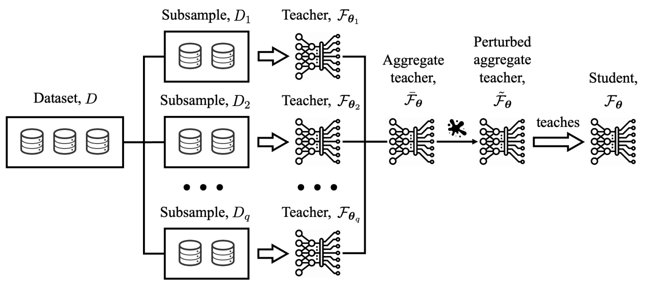
4.3 Private Aggregation of Teacher Ensembles

In Sec. 4.2 we have discussed that NN models are complex and the sensitivity is hard to quantify. Similar to what is done for KMC models in Sec. 3.4, one can again make the sensitivity easier to bound by sampling of the training data $D_1, D_2, \cdots, D_q$ and train NN models using each subset. Motivated by this, Papernot *et al.* [30] propose a general framework called ***Private Aggregation of Teacher Ensembles*** (PATE) to achieve DP.

Fig. 4 shows the design of PATE framework. The training dataset D is first randomly partitioned into disjoint subsets $D_1, D_2, ..., D_q$, and each subset is used to train a ***teacher*** model $\mathcal{F}_{\boldsymbol{\theta}_1}, \mathcal{F}_{\boldsymbol{\theta}_2}, ..., \mathcal{F}_{\boldsymbol{\theta}_q}$. Aggregation of these q teachers are done through majority vote: given a new set of features $\mathbf{x}$, each teacher will predict a label y, with q_c of them predicting $y = c$ (recall that class $c \in \mathcal{C}$). When 1 datum in training dataset changes, only one of $D_1, D_2, ..., D_q$ will change (since they are disjoint). Thus, there will be at most 1 teacher changing its predicting and thus a maximum of 2 q_c's changing: one increases by 1 and the other decreases by 1. Thus the sensitivity for predicting the label to $\mathbf{x}$ is 2. Therefore, one can inject noises to each q_c to obtain the perturbed aggregate teacher:

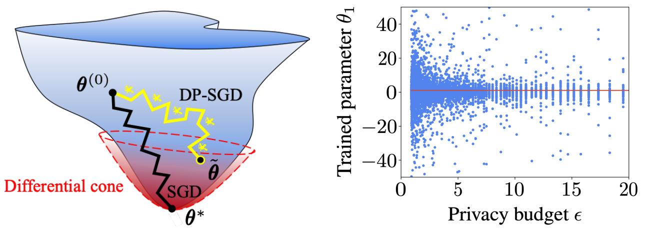
$$\tilde{q}_c \leftarrow q_c + \text{Lap}\left(\frac{2}{\epsilon}\right), \forall c \in \mathcal{C}. \tag{39}$$

The above mechanism satisfies ϵ-DP for the particular query $\mathbf{x}$. It does not mean that the perturbed aggregate teacher itself satisfies ϵ-DP, but rather mean that a privacy of ϵ is cost



Dataset, D
Subsample, D_1
Subsample, D_2
Subsample, D_q
Teacher, $\mathcal{F}_{\theta_1}$
Teacher, $\mathcal{F}_{\theta_2}$
Teacher, $\mathcal{F}_{\theta_q}$
Aggregate teacher, $\bar{\mathcal{F}}_{\theta}$
Perturbed aggregate teacher, $\tilde{\mathcal{F}}_{\theta}$
teaches
Student, $\mathcal{F}_{\theta}$

Figure 4: **Illustration of the PATE framework.**



(a) Differential cone. (b) Experiment.

Figure 5: **Visualization of the privacy-utility tradeoff.** DP-SGD will stop at somewhere close to, but not exactly at, the minimum (5a). It will be more likely to be accurate (i.e., has smaller variance) when the data owners choose a larger private budget ϵ.

whenever the perturbed aggregate teacher predicts the label to a new query.

The next stage of PATE is for a ***student*** model to learn from the perturbed aggregate teacher. To do this, the perturbed aggregate teacher needs to predict the label to T new queries in T iterations. These queries can either be public unlabelled dataset or synthetic feature set. The student model $\mathcal{F}_{\boldsymbol{\theta}}$ can therefore be trained using the newly labelled dataset. In other words, PATE is essentially an `InPert` mechanism where the perturbed aggregate teacher model creates the synthetic input dataset. The overall privacy level of the PATE framework can be analyzed through the aforementioned moment accountant method [1] as well.

The PATE framework has been shown to have a performance comparable to, if not better than, DP-SGD. For example, on the standard MNIST dataset, PATE has achieved both better privacy level ($\epsilon = 2.04$) and accuracy (98%) than DP-SGD ($\epsilon = 8$, 97% accuracy). Moreover, PATE does not restrict the type of ML/DL model used and the training mechanism, unlike DP-SGD. Papernot *et al.* [30] have successfully applied the PATE framework on GAN and random forest models. Nevertheless, the PATE framework can only be applied to classification problems so far (to let the sensitivity be 2). It remains questionable whether the framework can be adapted to regression problems as DP-SGD is capable of.

5 Discussions

In this section, we will make miscellaneous discussions about DP-ML methods and their implications.

5.1 Privacy-Utility Tradeoff

All DP-ML techniques inject noises at different ML stages to guarantee the privacy of data owners. The noises added inevitably affect the model utility/performance (e.g., accuracy). Lower privacy budget (i.e., smaller ϵ) leads to a worse utility. Such intuitions have also been theoretically proven by Alvim *et al.* [2], who state that a mechanism that provides ϵ-DP induces a bound on the utility. Thus, we are interested in finding out the tradeoff between privacy and utility, i.e., (a) what is the best utility that can be possibly achieved given a fixed privacy budget, and (b) maximally how much privacy budget can be guaranteed given a desired utility. In particular, (a) has been extensively studied in the field of DP-ML in order to achieve a model as good as possible; (b) has been rigorously studied as *privacy accounting*, which aims to provide tighter privacy bound for given DP-ML mechanisms. Besides, since the data of each data owner are different, the difference in trained model parameters $\boldsymbol{\theta}$ using two neighboring datasets which differ by different data owners' datum is also different. Therefore, the privacy level that a given DP-ML mechanism provides for each data owner is different too. Such works are known as *individualized privacy accounting* (e.g., [20]).

To visualize the privacy-utility tradeoff, we take DP-SGD as an example (Fig. 5). As shown in Fig. 5a, the path taken for $\boldsymbol{\theta}$ to converge is more noisy and thus the convergence is slower in DP-SGD than in normal SGD. Moreover, instead of landing at the minimum, DP-SGD always lands at some point inside a *differential cone* around the minimum $\boldsymbol{\theta}^*$. This is not only because the gradient becomes smaller as $\boldsymbol{\theta}$ is getting closer to the minimum $\boldsymbol{\theta}^*$ and will eventually be fully perturbed by the injected noises, but also because DP-SGD can only take a finite number of iterations. To illustrate the privacy-utility tradeoff, we also conduct a simple experiment as shown in Fig. 5b. Specifically, we train a linear model under different privacy budgets and plot the value of one particular entry θ_1 of the trained model parameters $\boldsymbol{\theta}$. The variance of points at each privacy budget corresponds to the size of differential cone. It can be seen clearly that when a smaller privacy budget ϵ is chosen, the variance is larger, which means that the model utility is more likely to be lower.

5.2 Privacy-Fairness Tradeoff

In ML, the dataset D used for model training is sometimes a combination of data from different groups (e.g., gender, race) or different classes. Due to the data collection procedures or distribution of different classes within access, the proportion of each group in dataset D can often be imbalanced. It is important that the trained ML model not only achieves a high overall utility, but also a high utility for each of the groups. ***Fairness*** studies how disparate the model utility for each group is. For example, consider a certain disease where 90% of the patients have mild symptoms and the rest of them have severe symptoms. A trained ML model may report that a certain drug is effective to 91% of patients, which seems good enough. However, when broken down to groups, it may be effective to 99% of patients with mild symptoms but only 20% of patients with severe symptoms. Therefore, it is not appropriate to advise those with severe symptoms to take this drug just because of the 91% overall accuracy. In other words, the trained ML model is biased towards those with mild symptoms and thus unfair.

Unfortunately, DP-ML could also lead to unfairness in the trained model. Consider a fair ML model which can perfectly represent both majority and minority groups. For the minority groups to have comparable impact on the trained model, the training mechanism must place higher importance on these data. However, in view of DP-ML, higher importance on particular data would raise the overall sensitivity. While DP aims to bound sensitivity, it will try to weaken the impact of these data and therefore be biased against them as before.

Take the two DP mechanisms in DL as an example. For DP-SGD, data from minority group will normally have a larger gradient because they are seldom learned in the training process. However, by clipping the gradient to a maximum bound G, the impact of such data is limited and therefore the trained model will be unfair. In PATE, the "anomalous" data will be even less frequent in each of the sampled subset to train the teacher models, and thus the aggregate teacher itself will be unfair. Student model that learns from the perturbed aggregate teacher will also be unfair. Therefore, the tradeoff between privacy and fairness is a trending research topic [3, 31]. For example, some research [36] has shown that DP-SGD has a more severe disparate impact than PATE.

5.3 Global and Local Differential Privacy

The DP-ML mechanisms introduced in this paper require the model owner to inject noise to the ML model throughout the training process. This demands that the model owner has full access to the training dataset and therefore should be trusted by data owners. However, as raised in Sec. 1, in some cases data owners do not trust that the model owner will keep their data confidential (will not **directly** release their data) and prefer a more private ML setting called federated learning (FL).

In FL, the model owner does not have access to the training data. Instead, model training is done locally by each data owner themselves. Each data owner will report only the trained model parameters using their data to the model owner. The model owner will then aggregate these model parameters as the final model parameters. Since FL ensures data confidentiality and is easy to implement, it has been widely used in industries involving sensitive data such as medicine [12].

However, if data can be leaked from trained ML model parameters, the untrusted model owner can also reconstruct each data owner's data, which defeats the purpose of FL. Therefore, the idea of DP should also be cultivated into FL such that the model owner could not differentiate the data owners from their reported model parameters. Out of this motivation, ***local differential privacy*** [39] is introduced as follows:

Definition 4. [Local Differential Privacy [39]] Given data owners $D_N = \{d_1, d_2, \cdots, d_N\}$, a federated randomized mechanism $\mathcal{M}_{\text{fed}} : D_N \to \mathbf{\Theta}$ satisfies *ϵ-local differential privacy* (ϵ-LDP) if for any pair of data owners $d, d' \in D_N$ and for any $O \subseteq \mathbf{\Theta}$, it holds that

$$\Pr(\mathcal{M}_{\text{fed}}(d) \in O) \leq e^{\epsilon} \cdot \Pr(\mathcal{M}_{\text{fed}}(d') \in O). \tag{40}$$

The mechanisms to achieve LDP are similar to those for global DP, such as the Laplacian mechanism for ϵ-LDP and the Gaussian mechanism for (ϵ, δ)-LDP which adds the tolerance δ to Eq. 4.

Concluding Remarks

Currently, the significance of DP-ML continues to expand. Laws such as General Data Protection Regulations (GDPR) have enforced *data ownership* and data owners have increasing control over their data. Large corporations have been deploying their own DP-ML mechanisms, such as Google's RAPPOR [18], Apple's DP deployments in its emojis, health and Safari browser [35], and Microsoft's DP mechanisms for telemetry data [14]. In this paper, we have thoroughly covered the basics and some advanced knowledge about the field of DP-ML. Although both academia and industries have made great efforts in the development and applications of DP-ML, there are still open challenges awaiting exploration.

References

[1] Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H. B.; Mironov, I.; Talwar, K.; and Zhang, L. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS'16. ACM.

[2] Alvim, M. S.; Andrés, M. E.; Chatzikokolakis, K.; Degano, P.; and Palamidessi, C. 2012. Differential Privacy: On the Trade-Off Between Utility and Information Leakage. In Barthe, G.; Datta, A.; and Etalle, S., eds., *Formal Aspects of Security and Trust*, 39–54. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-29420-4.

[3] Bagdasaryan, E.; Poursaeed, O.; and Shmatikov, V. 2019. Differential Privacy Has Disparate Impact on Model Accuracy. *Advances in neural information processing systems*, 32.

[4] Balle, B.; Barthe, G.; and Gaboardi, M. 2018. Privacy Amplification by Subsampling: Tight Analyses via Couplings and Divergences. *Advances in neural information processing systems*, 31.

[5] Bassily, R.; Smith, A.; and Thakurta, A. 2014. Private Empirical Risk Minimization: Efficient Algorithms and Tight Error Bounds. In *2014 IEEE 55th annual symposium on foundations of computer science*, 464–473. IEEE.

[6] Berkson, J. 1930. Bayes' Theorem. *The Annals of Mathematical Statistics*, 1(1): 42–56.

[7] Blum, A.; Dwork, C.; McSherry, F.; and Nissim, K. 2005. Practical Privacy: The SuLQ Framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 128–138.

[8] Chaudhuri, K.; and Monteleoni, C. 2008. Privacy-Preserving Logistic Regression. In *Proc. NeurIPS*, volume 21.

[9] Chaudhuri, K.; Monteleoni, C.; and Sarwate, A. D. 2011. Differentially Private Empirical Risk Minimization. *Journal of Machine Learning Research*, 12(3).

[10] Chaudhuri, K.; Sarwate, A. D.; and Sinha, K. 2013. Near-Optimal Algorithms for Differentially-Private Principal Components. arXiv:1207.2812.

[11] Chikuse, Y. 2012. *Statistics on Special Manifolds*, volume 174. Springer Science & Business Media.

[12] Dayan, I.; Roth, H. R.; Zhong, A.; Harouni, A.; Gentili, A.; Abidin, A. Z.; Liu, A.; Costa, A. B.; Wood, B. J.; Tsai, C.-S.; et al. 2021. Federated Learning for Predicting Clinical Outcomes in Patients With COVID-19. *Nature medicine*, 27(10): 1735–1743.

[13] Desfontaines, D.; and Pejó, B. 2022. SoK: Differential Privacies. arXiv:1906.01337.

[14] Ding, B.; Kulkarni, J.; and Yekhanin, S. 2017. Collecting Telemetry Data Privately. *Advances in Neural Information Processing Systems*, 30.

[15] Dwork, C. 2006. Differential Privacy. In *International Colloquium on Automata, Languages, and Programming*, 1–12.

[16] Dwork, C.; Kenthapadi, K.; McSherry, F.; Mironov, I.; and Naor, M. 2006. Our Data, Ourselves: Privacy via Distributed Noise Generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, 486–503. Springer.

[17] Dwork, C.; and Roth, A. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4): 211–407.

[18] Erlingsson, Ú.; Pihur, V.; and Korolova, A. 2014. Rappor: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 1054–1067.

[19] Fan, W.; Wang, H.; Yu, P. S.; and Ma, S. 2003. Is Random Model Better? On Its Accuracy and Efficiency. In *Third IEEE International Conference on Data Mining*, 51–58. IEEE.

[20] Feldman, V.; and Zrnic, T. 2021. Individual Privacy Accounting via a Rényi Filter. *Advances in Neural Information Processing Systems*, 34: 28080–28091.

[21] Gillespie, R. 1955. Principles of Mathematical Analysis. By Walter Rudin. Pp. x, 227. 40s. 1953. (McGraw-Hill) - Theory of Functions of Real Variable. By Henry P. Thielman. Pp. xiv, 209. 35s. 1953. (Butterworth Scientific Publications, London). *The Mathematical Gazette*, 39(329): 258–259.

[22] Gong, M.; Xie, Y.; Pan, K.; Feng, K.; and Qin, A. K. 2020. A Survey on Differentially Private Machine Learning. *IEEE computational intelligence magazine*, 15(2): 49–64.

[23] Jagannathan, G.; Pillaipakkamnatt, K.; and Wright, R. N. 2009. A Practical Differentially Private Random Decision Tree Classifier. In *2009 IEEE International Conference on Data Mining Workshops*, 114–121.

[24] Kamath, G. 2020. Lecture 7 — Exponential Mechanism. http://www.gautamkamath.com/CS860notes/lec7.pdf. Accessed: 2024-04-04.

[25] LeCun, Y.; Touresky, D.; Hinton, G.; and Sejnowski, T. 1988. A Theoretical Framework for Back-Propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, 21–28.

[26] Lei, J. 2011. Differentially Private M-Estimators. In *Advances in neural information processing systems*, volume 24.

[27] Li, L.; Fan, Y.; Tse, M.; and Lin, K.-Y. 2020. A Review of Applications in Federated Learning. *Computers & Industrial Engineering*, 149: 106854.

[28] McSherry, F.; and Talwar, K. 2007. Mechanism Design via Differential Privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, 94–103. IEEE.

[29] Nissim, K.; Raskhodnikova, S.; and Smith, A. 2007. Smooth Sensitivity and Sampling in Private Data Analysis. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, 75–84. New York, NY, USA: Association for Computing Machinery. ISBN 9781595936318.

[30] Papernot, N.; Abadi, M.; Úlfar Erlingsson; Goodfellow, I.; and Talwar, K. 2017. Semi-Supervised Knowledge Transfer for Deep Learning From Private Training Data. arXiv:1610.05755.

[31] Rosenblatt, L.; Stoyanovich, J.; and Musco, C. 2024. A Simple and Practical Method for Reducing the Disparate Impact of Differential Privacy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 21554–21562.

[32] Shamir, O.; and Zhang, T. 2013. Stochastic Gradient Descent for Non-Smooth Optimization: Convergence Results and Optimal Averaging Schemes. In *International conference on machine learning*, 71–79. PMLR.

[33] Shokri, R.; Stronati, M.; Song, C.; and Shmatikov, V. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*, 3–18. IEEE.

[34] Sweeney, L. 2000. Simple Demographics Often Identify People Uniquely. *Carnegie Mellon University, Data Privacy*.

[35] Team, D. P. 2017. Learning with Privacy at Scale Differential.

[36] Uniyal, A.; Naidu, R.; Kotti, S.; Singh, S.; Kenfack, P. J.; Mireshghallah, F.; and Trask, A. 2022. DP-SGD vs PATE: Which Has Less Disparate Impact on Model Accuracy? arXiv:2106.12576.

[37] Vaidya, J.; Shafiq, B.; Basu, A.; and Hong, Y. 2013. Differentially Private Naive Bayes Classification. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, 571–576.

[38] Wang, D.; Ye, M.; and Xu, J. 2017. Differentially Private Empirical Risk Minimization Revisited: Faster and More General. *Advances in Neural Information Processing Systems*, 30.

[39] Yang, M.; Guo, T.; Zhu, T.; Tjuawinata, I.; Zhao, J.; and Lam, K.-Y. 2023. Local Differential Privacy and Its Applications: A Comprehensive Survey. *Computer Standards & Interfaces*, 103827.

[40] Zhang, J.; Zhang, Z.; Xiao, X.; Yang, Y.; and Winslett, M. 2012. Functional Mechanism: Regression Analysis under Differential Privacy. In *Proceedings of the VLDB Endowment*, volume 5.

A Project Administration

A.1 Contributions

Overall, the two authors have contributed equally to the report.

A.2 Acknowledgement

The authors would like to extend their thanks to Assistant Professor Yong Sheng Soh for his guidance throughout the course *MA4270: Data Modelling and Computation* at National University of Singapore.

B Additional Explanations and Examples of ML and DP-ML Models

B.1 Decision Tree Models

In this section, we will give an example of IDT using IG as the selection criterion.

Suppose the dataset D is as shown below, where each row is a datum and the last column FINALGRADE is the class label to be predicted.

	ATTENDLECTURES	ATTENDTUTORIALS	MIDTERMGRADE	FINALGRADE
$\mathbf{x}_1$	Always	Always	Pass	Pass
$\mathbf{x}_2$	Always	Sometimes	Fail	Pass
$\mathbf{x}_3$	Sometimes	Rarely	Pass	Pass
$\mathbf{x}_4$	Sometimes	Rarely	Fail	Fail
$\mathbf{x}_5$	Rarely	Sometimes	Fail	Fail
$\mathbf{x}_6$	Rarely	Rarely	Pass	Pass

To construct a DT model, we can first calculate[6] the entropy in the entire dataset D:

$$\begin{aligned}\text{Entropy}(D) &= -\Pr(\text{Pass})\log_2\Pr(\text{Pass}) - \Pr(\text{Pass})\log_2\Pr(\text{Pass})\\ &= -\frac{4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right)\\ &= 0.918.\end{aligned}$$

For simplicity we define $h(a) = -a\log_2(a) - (1-a)\log_2(1-a)$ for $a \in [0,1]$, i.e., it is the entropy of a dataset with binary labels, where the fraction of one label is a and the other is $1-a$.

Next, we can compute the information gain by each attribute as follows

$$\begin{aligned}\text{IG}(\textsc{AttendLectures}) &= 0.918 - \frac{2}{6}h(1) - \frac{2}{6}h\left(\frac{1}{2}\right) - \frac{2}{6}h\left(\frac{1}{2}\right)\\ &= 0.918 - 0 - \frac{1}{3} - \frac{1}{3}\\ &= 0.252.\\ \text{IG}(\textsc{AttendTutorials}) &= 0.918 - \frac{1}{6}h(1) - \frac{2}{6}h\left(\frac{1}{2}\right) - \frac{3}{6}h\left(\frac{2}{3}\right)\\ &= 0.918 - 0 - \frac{1}{3} - 0.459\\ &= 0.126.\\ \text{IG}(\textsc{MidtermGrade}) &= 0.918 - \frac{3}{6}h\left(1\right) - \frac{3}{6}h\left(\frac{1}{3}\right)\\ &= 0.918 - 0 - 0.459\\ &= 0.459.\end{aligned}$$

Since the attribute MIDTERMGRADE provides the highest information gain, this attribute is chosen as the root node. This process is repeated at child nodes where data have different class labels until all attributes have been queried, forming the DT model shown in Fig. 6.

[6]Note that since there are two classes, $\log_2$ is used so that entropy is in the range $[0, 1]$. Entropy equals 0 when all data belong to the same class, i.e., no uncertainty about the classification. Entropy is 1 when class labels are half-half.

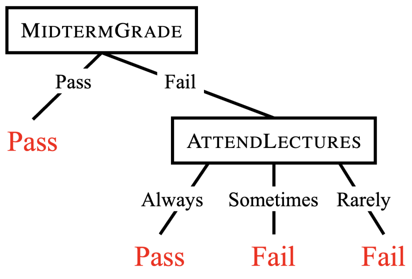


Figure 6: **Example of a trained DT model.**

B.2 Neural Network Models

In this section we will give an example of backpropagation.

Suppose we are given a dataset with only one datum with features $\mathbf{x} = (1, 2)$ and label $y = 1$. We want to build an NN model with one hidden layer that consists of two perceptrons (Fig. 7) and no bias term θ_0. Model training starts with randomly initialized parameters $\boldsymbol{\theta}^{(0)} = (\theta_1 = 0.10, \theta_2 = 0.20, \theta_3 = 0.05, \theta_4 = 0.15, \theta_5 = 0.08, \theta_6 = 0.12)$.

We first use $\boldsymbol{\theta}^{(0)}$ and inputs to predict the label. Inputs are multiplied by weights and passed forward to the next layer:

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0.10 & 0.20 \\ 0.05 & 0.15 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.50 \\ 0.35 \end{bmatrix};$$

$$\text{Predictions } \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \begin{bmatrix} 0.08 & 0.12 \end{bmatrix} \begin{bmatrix} 0.50 \\ 0.35 \end{bmatrix} = 0.082.$$

Now we can calculate the squared loss $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ (the factor $\frac{1}{2}$ is added to ease the calculation):

$$\mathcal{L}(0.082, 1) = \frac{1}{2}(0.082 - 1)^2 = 0.421.$$

To update the weights according to the error, we use partial derivative with respect to each weight and apply chain rule. We start from the last layer:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta_6} &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial \theta_6} \\ &= \frac{\partial \frac{1}{2}(\hat{y} - y)^2}{\partial \hat{y}} \times \frac{\partial (z_1\theta_5 + z_2\theta_6)}{\partial \theta_6} \\ &= (\hat{y} - y) \times \frac{\partial (\hat{y} - y)}{\partial \hat{y}} \times z_2 \\ &= (\hat{y} - y) z_2. \end{aligned}$$

So θ_6 can be updated by $\theta_6 \leftarrow \theta_6 - \eta(\hat{y} - y)z_2$ for some positive learning rate η (Eq. 37). Similarly, we have $\frac{\partial \mathcal{L}}{\theta_5} = (\hat{y} - y)z_1$ and $\theta_5 \leftarrow \theta_5 - \eta(\hat{y} - y)z_1$. Now we can compute the partial derivative for the second last layer:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\theta_1} &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1} \\ &= (\hat{y} - y) \times \frac{\partial \left(z_1\theta_5 + z_2\theta_6\right)}{\partial z_1} \times \frac{\partial x_1\theta_1 + x_2\theta_2}{\partial \theta_1} \\ &= (\hat{y} - y)\theta_5 x_1. \end{aligned}$$

We can find the formula to update $\theta_2, \theta_3, \theta_4$ and θ_5 accordingly. Suppose the learning rate is $\eta = 0.05$, after substituting in the values, we have $\theta_1 = 0.104, \theta_2 = 0.207, \theta_3 = 0.056, \theta_4 = 0.161, \theta_5 = 0.103$ and $\theta_6 = 0.136$. The new prediction is thus

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0.104 & 0.207 \\ 0.056 & 0.161 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.518 \\ 0.378 \end{bmatrix};$$

$$\text{Predictions } \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \begin{bmatrix} 0.103 & 0.136 \end{bmatrix} \begin{bmatrix} 0.518 \\ 0.378 \end{bmatrix} = 0.105,$$

which is closer to the actual label as compared to the previous prediction. This process continues until the loss is minimized and all the partial derivatives equal zero.

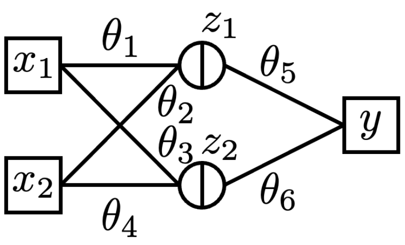
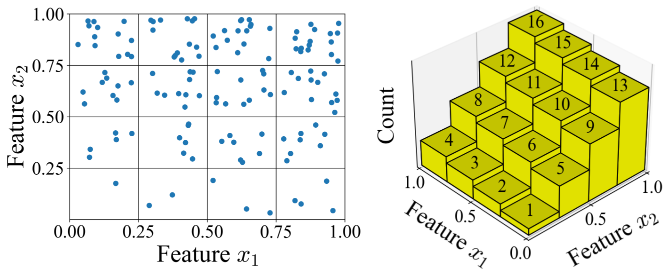


Figure 7: **Example of an NN model.**



(a) Dataset with 2 features x_1, x_2. (b) Histogram.

Figure 8: **Example of a dataset's histogram which reflects its density function.**

B.3 Differentially Private M-Estimators

M-estimator is a family of statistical estimators including commonly used ones such as least squares estimators (which can be used for LinR) and maximum likelihood estimators (which can be used for LogR). The computation of M-estimators requires post-processing of the dataset D's *density function*, that is, the proportion of data in each cubic cell of dataset D. This can be easily found from the *histogram* of D (Fig. 8). In any two neighboring datasets, the numbers of data in the same cubic cell differ by at most 1 because only 1 datum changes. Therefore, to achieve ϵ-DP, one can simply add a $\mathrm{Lap}(1/\epsilon)$ noise to each cubic cell of the histogram. Every M-estimator computed from any synthetic dataset that follows the perturbed histogram then satisfies ϵ-DP due to its post-processing robustness property (Prop. 2).