

Differential Privacy in Deep Learning

Ref:

Deep Learning with Differential Privacy

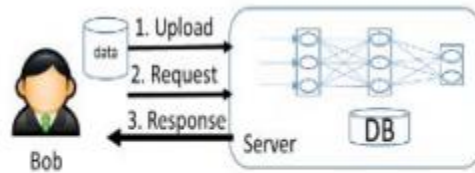
[<https://arxiv.org/pdf/1607.00133.pdf>]

Differential Privacy in Deep Learning: An Overview

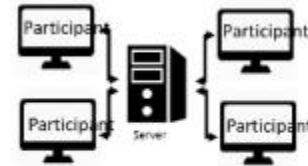
[<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9044259&tag=1>]

Depending on the models of supervised, unsupervised and semi-supervised learning as well as the learning model organization used for solving problems in deep learning, two types of learning model :

- Centralized learning
- decentralized learning (collaborative learning)



a. Centralized learning



b. Decentralized learning

Centralized learning

- Server collects data from user to build the model.
- Users lose control over the data after uploading personal data to server in many cases
- user privacy is not safety guaranteed.
- Server can infer sensitive information from user's data, or the adversary attacks server to steal the personal information of user.

Collaborative learning

- each client (individual, organization) in collaborative learning trains a batch locally.
- Then receive the gradient from server and calculates the gradient that is applied to its weights to minimize the cost function.
- All the clients or a subset of them send their gradients to the parameter server, which reads all the client's gradients, optimizes them and updates the model stored in the parameter server.

Collaborative learning

Collaborative learning is also based on the idea of collecting and exchanging parameters, which code the raw data but it has the specific characteristics of the client's dataset.

This is a threat which is presented by Generative Adversarial Networks (GANs) [17], specially classified images into categories. GANs are trained to generate similar-looking sample to the training set, without actually having access to the victim data.

Two privacy preserving method:

differential privacy and homomorphic encryption

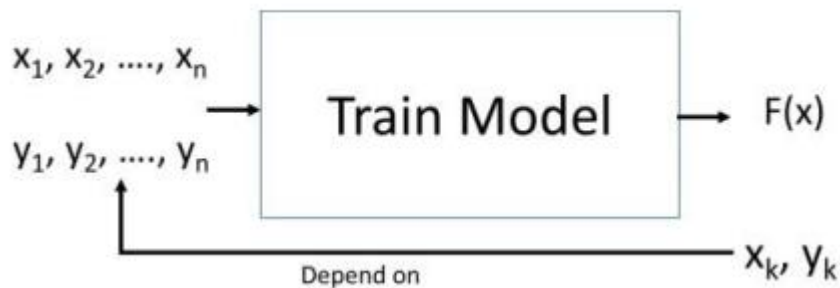
(homomorphic encryption which is an encryption method, permit arbitrary computations on encrypted data without decrypting it.).

Prevention of Possible Attacks with Differential Privacy Framework

Inference Attack

Inference attacks in deep learning fall into two fundamental categories:

- reconstruction attacks
- Tracing (membership inference) attacks

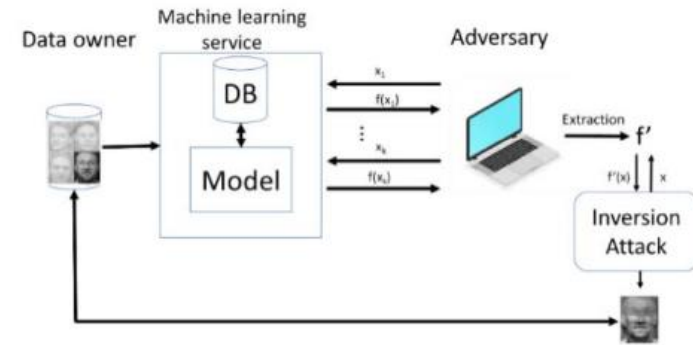


Reconstruction attacks

- The attacker's objective is to extract training data from output model predictions.
- The constructed model inversion attacks for deep models use the output of the model to infer certain features of the training set.
- Especially, in the facial recognition, [10 in Ref paper] Fredrikson's research has shown that training data can be reconstructed from the model.

Model inversion attacks

- Features synthesized from the model to generate an input that maximizes the likelihood of being predicted with a certain label.
- The adversary's objective is to train a substitute model F' , that is capable of mimicking a target model F .
- To build model F' , it is based on leakage of information that is implemented in the extraction time.
- In model extraction, the adversary only has to access to the prediction API of a target model and query the target model iterative using "natural" or synthetic samples.
- These samples are specifically crafted to maximize the extraction of information about the model internals from the predictions returned by the model F .



Tracing attacks [black-box access]

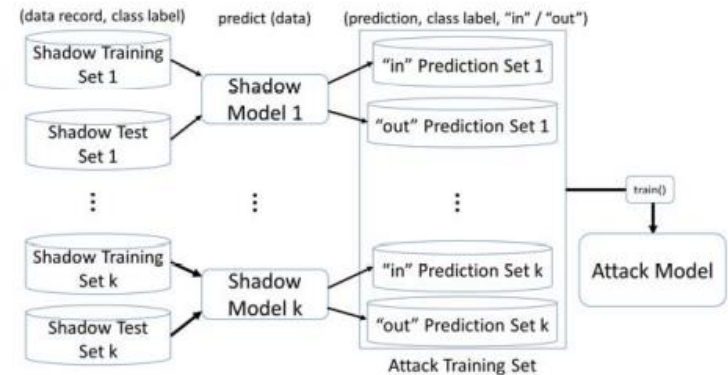
In the first step, the adversary queries the target record t and uses the target classifiers' predictions on t to infer the membership status of t .

For each record t , there are two possible classes: class label “in”, which means that the record is in the training set, and class label “out”, which represents that the record is not in the training set.

For the next step, the “shadow” training technique is built to use for the membership classification task.

Multiple “shadow models” that are trained by the adversary use the same machine learning algorithm on records sampled from the data in the first step.

These shadow models are used to simulate the behavior of the target model and generate



Privacy threats

Central system:

- Companies provide deep learning models and services to the public such as machine learning as a service like Microsoft Azure Learning, Google, Amazon, BigML, etc.
- Data owner send data to server and public model service to client.
- The adversary sends an input to public machine learning service and receives an output.
- Using the inputs and outputs pair, the attacker can train his own local model which is similar to the target model.

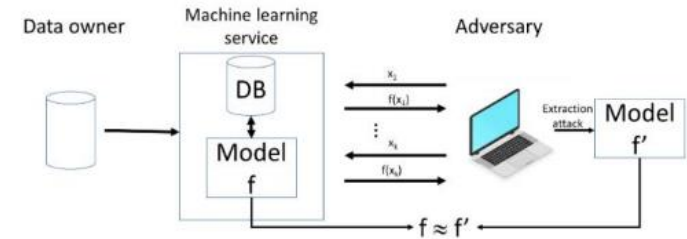
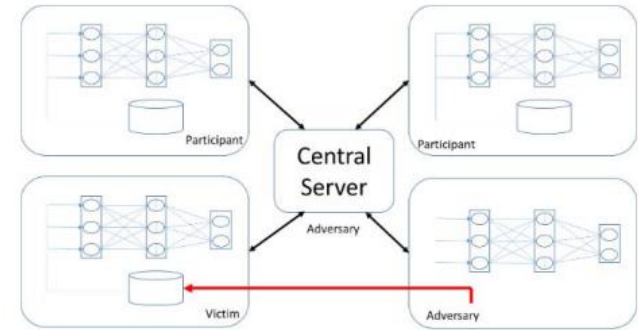


Fig. 7. The model extract attack in central learning system.

Privacy threats

In collaborative learning system:

- Clients train a batch locally, and then calculates the gradient that is applied to its weights to minimize the cost function.
- Finally, it sends the gradient to the parameter server.
- If the adversary has taken the role of model server, they receive the client's gradient. This is called “model steal attack”.



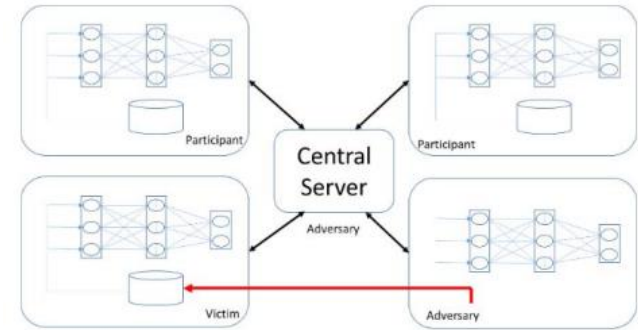
The threat in collaborative learning system.

Privacy threats

In collaborative learning system:

If the adversary has taken the role of participant, who attack to steal other participant information from the training set.

This attack based on exploiting the real-time nature of model learning, which allows the adversary to train a GAN that generates prototypical samples of the private training set.



The threat in collaborative learning system.

DEFENSES BY DIFFERENTIAL PRIVACY IN DEEP LEARNING

Objective

- In ML, we manage sensitive database and would like to release some statistics from this data to the public.
- However, we have to ensure that it's impossible for an adversary to reverse-engineer the sensitive data from what we've released.
- An adversary in this case is a party with the intent to reveal, or to learn, at least some of our sensitive data.
- This reverse-engineering is a type of privacy breach.
- **Anonymizing the data will not be good enough**

Differentially-private algorithms

These algorithms rely on incorporating random noise into the mix so that everything an adversary receives becomes noisy and imprecise, and so it is much more difficult to breach privacy (if it is feasible at all).

Rating	Count
Bad	3
Normal	1510
Good	200

Table 1: A summary of a sensitive database.

Query Number	Response
1	2.915
2	1.882
3	1.292
4	4.026
5	5.346
Average	3.090
90% confidence interval	1.696 to 4.484

Table 2: Simulated responses to a queried answer.

For this example, let's assume that the adversary wants to know the number of people who have a bad credit rating (which is 3). The data is sensitive, so we cannot reveal the ground truth. Instead we will use an algorithm that returns the ground truth, $N = 3$, plus some random noise.

This basic idea (adding random noise to the ground truth) is key to differential privacy.

Rating	Count
Bad	3
Normal	1510
Good	200

Table 1: A summary of a sensitive database.

Let's say we choose a random number L from a zero-centered Laplace distribution with standard deviation of 2.

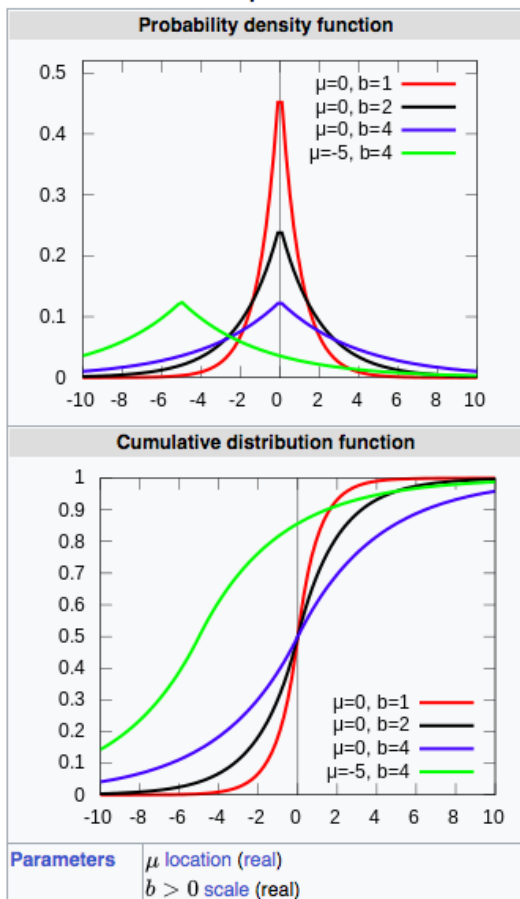
We return $N+L$.

This algorithm is “noisy counting”.

Query Number	Response
1	2.915
2	1.882
3	1.292
4	4.026
5	5.346
Average	3.090
90% confidence interval	1.696 to 4.484

Table 2: Simulated responses to a queried answer.

Laplace



PDF	$\frac{1}{2b} \exp\left(-\frac{ x-\mu }{b}\right)$
CDF	$\begin{cases} \frac{1}{2} \exp\left(\frac{x-\mu}{b}\right) & \text{if } x \leq \mu \\ 1 - \frac{1}{2} \exp\left(-\frac{x-\mu}{b}\right) & \text{if } x \geq \mu \end{cases}$

How much noise to add?

If we're going to report a nonsensical, noisy answer, what is the point of this algorithm? The answer is that it allows us to strike a balance between utility and privacy. We want the general public to gain *some* utility from our database, without exposing *any* sensitive data to a possible adversary.

- The first answer the adversary receives is close to, but not equal to, the ground truth. In that sense, the adversary is fooled, utility is provided, and privacy is protected.

- we increase the variance of the added noise, we can better defend our data, but we can't simply add any Laplace noise to a function we want to conceal.
- **the amount of noise we must add depends on the function's sensitivity, and each function has its own sensitivity.**

The sensitivity of a function is the largest possible difference that one row can have on the result of that function, for *any* dataset.

• Mechanism

Example: .

- This function has a sensitivity of 5, and we can show this quite easily. If we have two datasets of size 10 and 11, they differ in one row, but the “counts rounded up to a multiple of 5” are 10 and 15 respectively. In fact, five is the largest possible difference this example function can produce. Therefore, its sensitivity is 5.
- Calculating the sensitivity for an arbitrary function is difficult

The sensitivity of a function is the largest possible difference that one row can have on the result of that function, for *any* dataset.

The Mechanism

For a function:

$$f: \mathcal{D} \mapsto \mathcal{R}^k$$

the **sensitivity** of f is:

$$\Delta f = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_1$$

on datasets D_1, D_2 differing on at most one element [1].

How does sensitivity relate to noisy counting?

How much noise to add?

- The larger the sensitivity S , the noisier the answer will be.

Input: Function F , Input X , Privacy level ϵ

Output: A Noisy Answer

Compute $F(X)$

Compute sensitivity of $F : S$

Draw noise L from a Laplace distribution with variance:

$$2 * \left(\frac{S}{\epsilon}\right)^2$$

Return $F(X) + L$

Limitation

- The first answer the adversary receives is close to, but not equal to, the ground truth. In that sense, the adversary is fooled, utility is provided, and privacy is protected.
- But after multiple attempts, they will be able to estimate the ground truth. This “estimation from repeated queries” is one of the fundamental limitations of differential privacy.
- If the adversary can make enough queries to a differentially-private database, they can still estimate the sensitive data. In other words, with more and more queries, privacy is breached. Table [2](#) empirically demonstrates this privacy breach.

Query Number	Response
1	2.915
2	1.882
3	1.292
4	4.026
5	5.346
Average	3.090
90% confidence interval	1.696 to 4.484

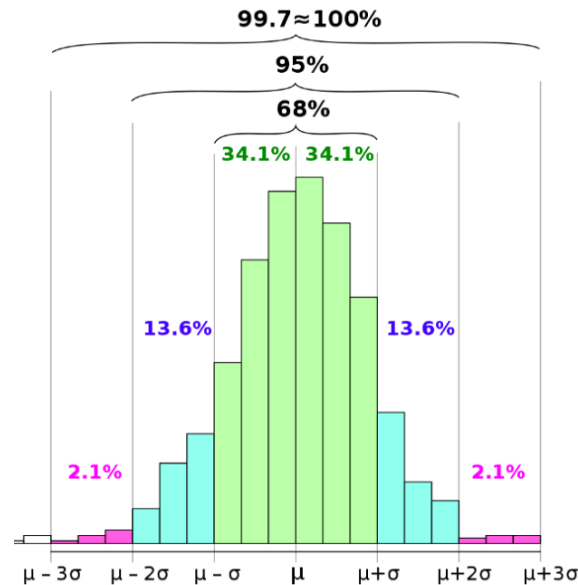
Table 2: Simulated responses to a queried answer.

- However, the 90% confidence interval from these 5 queries is quite wide; the estimate is not very accurate yet.

Confidence interval

- A confidence interval displays the probability that a parameter will fall between a pair of values around the mean.
- Confidence intervals measure the degree of uncertainty or certainty in a sampling method.

As expected, the confidence intervals become narrower as the adversary makes more queries (since the sample size increases, and the noise averages to zero).



Parameter epsilon: privacy loss

- Experiment shows: noisy counting example had an epsilon of 0.707, which is quite small. But still breached privacy after 50 queries.
- Because with increased queries, the **privacy budget** grows, and therefore the privacy guarantee is worse.

Input: Function F , Input X , Privacy level ϵ

Output: A Noisy Answer

Compute $F(X)$

Compute sensitivity of $F : S$

Draw noise L from a Laplace distribution with variance:

$$2 * \left(\frac{S}{\epsilon}\right)^2$$

Return $F(X) + L$

Query Number	Response
1	2.915
2	1.882
3	1.292
4	4.026
5	5.346
Average	3.090
90% confidence interval	1.696 to 4.484

Table 2: Simulated responses to a queried answer.

Privacy losses accumulate

- When two answers are returned to an adversary, the total privacy loss is twice as large, and the privacy guarantee is half as strong.
- This cumulative property is a consequence of the *composition theorem*.
- In essence, with each new query, additional information about the sensitive data is released.

Composition theorem has a pessimistic view and assumes the worst-case scenario:

- the same amount of leakage happens with each new response.
- For strong privacy guarantees, we want the privacy loss to be small.

How does differential privacy work if the privacy loss grows so quickly?

- To ensure a meaningful privacy guarantee, data curators can enforce a *maximum* privacy loss.
- If the number of queries exceeds the threshold, then the privacy guarantee becomes too weak and the curator stops answering queries.
- **The maximum privacy loss is called the privacy budget.**
 - Each query as a privacy ‘expense’ which incurs an incremental privacy loss. The strategy of using budgets, expenses and losses is fittingly known as *privacy accounting*.
- **Privacy accounting is an effective strategy to compute the privacy loss after multiple queries, but it can still incorporate the composition theorem.**

Formal Definition

Think of a database as a set of rows. We say databases D_1 and D_2 *differ in at most one element* if one is a proper subset of the other and the larger database contains just one additional row.

Definition 1. A randomized function \mathcal{K} gives ϵ -differential privacy if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq \text{Range}(\mathcal{K})$,

$$\Pr[\mathcal{K}(D_1) \in S] \leq \exp(\epsilon) \times \Pr[\mathcal{K}(D_2) \in S] \quad (1)$$

Properties


Differential privacy has several properties that make it particularly useful in applications such as ours: composability, group privacy, and robustness to auxiliary information.

- Composability enables modular design of mechanisms: if all the components of a mechanism are differentially private, then so is their composition.
- Group privacy implies graceful degradation of privacy guarantees if datasets contain correlated inputs, such as the ones contributed by the same individual.
- Robustness to auxiliary information means that privacy guarantees are not affected by any side information available to the adversary

Basic composition theorem

- Sequential composition
- Parallel composition
- Post processing

Sequential Composition

The first major property of differential privacy is *sequential composition* , which bounds the total privacy cost of releasing multiple results of differentially private mechanisms on the same input data. Formally, the sequential composition theorem for differential privacy says that:

- If $F_1(x)$ satisfies ϵ_1 -differential privacy
- And $F_2(x)$ satisfies ϵ_2 -differential privacy
- Then the mechanism $G(x) = (F_1(x), F_2(x))$ which releases both results satisfies $\epsilon_1 + \epsilon_2$ -differential privacy

Sequential composition is a vital property of differential privacy because it enables the design of algorithms that consult the data more than once.

Sequential composition is also important when multiple separate analyses are performed on a single dataset, since it allows individuals to bound the *total* privacy cost they incur by participating in all of these analyses. The bound on privacy cost given by sequential composition is an *upper* bound - the actual privacy cost of two particular differentially private releases may be smaller than this, but never larger.

Parallel Composition

Parallel composition can be seen as an alternative to sequential composition - a second way to calculate a bound on the total privacy cost of multiple data releases.

Parallel composition is based on the idea of splitting your dataset into disjoint chunks and running a differentially private mechanism on each chunk separately.

Since the chunks are disjoint, each individual's data appears in *exactly* one chunk - so even if there are k chunks in total (and therefore k runs of the mechanism), the mechanism runs exactly once on the data of each *individual*.

- If $F(x)$ satisfies ϵ -differential privacy
- And we split a dataset X into k disjoint chunks such that $x_1 \cup \dots \cup x_k = X$
- Then the mechanism which releases all of the results $F(x_1), \dots, F(x_k)$ satisfies ϵ -differential privacy

Histograms

In our context, a *histogram* is an analysis of a dataset which splits the dataset into “bins” based on the value of one of the data attributes, and counts the number of rows in each bin. For example, a histogram might count the number of people in the dataset who achieved a particular educational level.

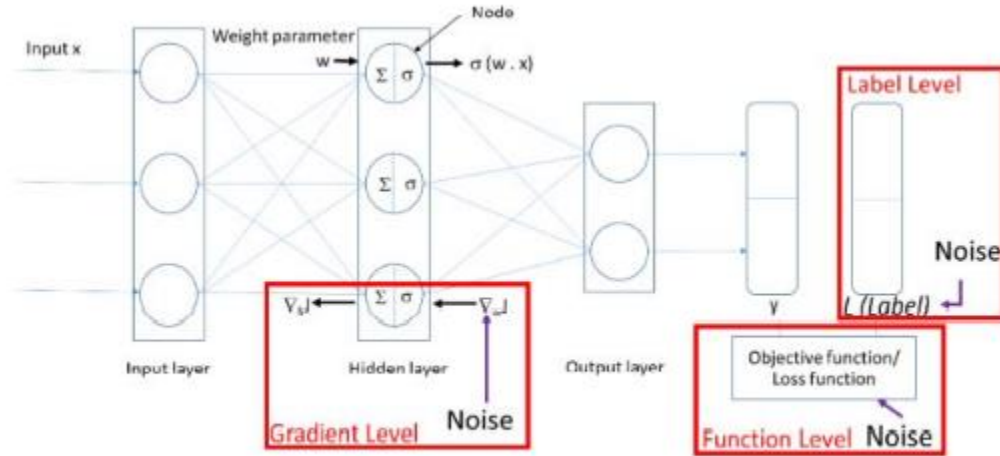
Contingency Tables

A *contingency table* or *cross tabulation* (often shortened to *crosstab*) is like a multi-dimensional histogram. It counts the frequency of rows in the dataset with particular values for more than one attribute at a time. Contingency tables are frequently used to show the relationship between two variables when analyzing data.

How much to add noise?

- Adding overly conservative noise to the parameters:
- Where the noise is selected according to the worst case analysis, would destroy the utility of the learned model.
- Proposed sophisticated approach to control the influence of the training data during the training process, specifically in the SGD computation.

Where to add the noise?



Protect privacy in deep learning model

Approaches: Gradient level, function-level, label-level

Revisit Gradient descent

Gradient descent (GD) is an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function.

Loss function:

Calculated using Mean Squared Error (MSE), squares the difference between every network output and true label, and takes the average.

C : loss function (also known as the *cost function*), N is the number of training images, \mathbf{y} is a vector of true labels ($\mathbf{y} = [target(\mathbf{x}_1), target(\mathbf{x}_2)...target(\mathbf{x}_n)]$), and \mathbf{o} is a vector of network predictions:

$$C(\mathbf{y}, \mathbf{o}) = \frac{1}{N} \sum_{i=1}^N (y_i - o_i)^2$$

Revisit Gradient descent

Batch gradient descent

Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters :

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta).$$

S Stochastic gradient descent (SGD) in contrast performs a parameter update for *each* training example $x^{(i)}$ and label $y^{(i)}$:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}).$$

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update.

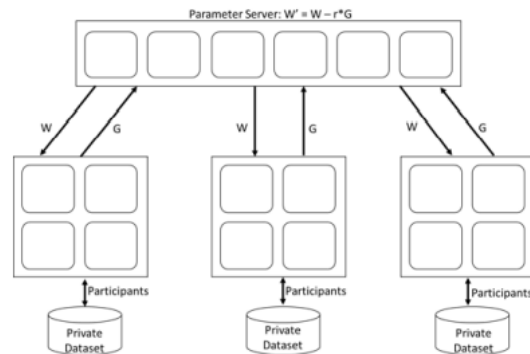
SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

Gradient in Collaborative Learning*

- The gradient level approach which injects noise into the gradients of the parameters in the client before sending to server, solve the issue in the collaborative learning.

How to achieve tighter estimation for many iterations and dynamic privacy budget allocation mechanism to improve the performance?

*Deep Learning with Differential Privacy, 2016,
<https://arxiv.org/pdf/1607.00133.pdf>



The collaborator learning architecture with Differential Private Stochastic Gradient Descent. W represents the parameter and G represents the gradient information

Differentially Private SGD Algorithm

- SGD implies **Stochastic Gradient Descent (SGD)**
- Gradient Descent is a popular optimization technique in Machine Learning and Deep Learning.
- [A gradient is the slope of a function. It measures the degree of change of a variable in response to the changes of another variable. Mathematically, Gradient Descent is a convex function whose output is the partial derivative of a set of parameters of its inputs. The greater the gradient, the steeper the slope.]
- **'stochastic' means a system or a process that is linked with a random probability.**
- **In SGD, since only one sample from the dataset is chosen at random for each iteration.**

Algorithm 1 Differentially private SGD (Outline)

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

Initialize θ_0 randomly

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

 Take a random sample L_t with sampling probability L/N

Compute gradient

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

Clip gradient

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

Add noise

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

Descent

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

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

- The pseudocode for Algorithm 1 groups all the parameters into a single input θ of the loss function L .
- For multi-layer neural networks, we consider each layer separately.
- Algorithm 1 estimates the gradient of L by computing the gradient of the loss on a group of examples and taking the average. This average provides an unbiased estimate

Algorithm 1 Differentially private SGD (Outline)

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

Initialize θ_0 randomly

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

 Take a random sample L_t with sampling probability L/N

Compute gradient

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

Clip gradient

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

Add noise

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

Descent

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

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

For ease of analysis:
 each lot is formed by
independently picking each
example with probability $q = L/N$,
where N is the size of the input
datas

Algorithm 1 Differentially private SGD (Outline)

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

Initialize θ_0 randomly

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

 Take a random sample L_t with sampling probability L/N

Compute gradient

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

Clip gradient

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

Add noise

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

Descent

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

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

Norm clipping: Proving the differential privacy guarantee of Algorithm 1 requires bounding the influence of each individual example on $\tilde{\mathbf{g}}_t$. Since there is no *a priori* bound on the size of the gradients, we *clip* each gradient in ℓ_2 norm; i.e., the gradient vector \mathbf{g} is replaced by $\mathbf{g} / \max(1, \frac{\|\mathbf{g}\|_2}{C})$, for a clipping threshold C . This clipping ensures that if $\|\mathbf{g}\|_2 \leq C$, then \mathbf{g} is preserved, whereas if $\|\mathbf{g}\|_2 > C$, it gets scaled down to be of norm C . We remark that gradient clipping of this form is a popular ingredient of SGD for deep networks for non-privacy reasons, though in that setting it usually suffices to clip after averaging.

Algorithm 1 Differentially private SGD (Outline)

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

Initialize θ_0 randomly

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

 Take a random sample L_t with sampling probability L/N

Compute gradient

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

Clip gradient

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

Add noise

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

Descent

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

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

Privacy accounting:

- For differentially private SGD, an important issue is computing the overall privacy cost of the training.
- The composability of differential privacy.

Limitation

- The magnitude of injected noise and the privacy budget are accumulated in proportion to the number of training epochs and the number of shared parameters.
- Thus, it may consume an unnecessarily large portion of the privacy budget as the number of training epochs and the number of shared parameters among multiple parties are often large.
- Abadi et al. [12] proposed a privacy accountant, which keeps track of privacy spending and enforces applicable privacy policies. However, the approach is still dependent on the number of training epochs, as it introduces noise into “gradients” of parameters in every training step.

Limitation

- Another drawback of the existing techniques is that all parameters are treated the same in terms of the amount of noise injected. This may not be ideal in real scenarios, since different features and parameters normally have different impacts upon the model output.

Other approaches: 1. Loss function*

- Motivated by this, Adaptive Laplace Mechanism (AdLM):
 - preserve differential privacy in deep learning.
 - intentionally add “more noise” into features which are “less relevant” to the model output, and vice-versa.
 - inject Laplace noise into the computation of Layer-wise Relevance Propagation (LRP) to estimate a differentially private relevance of each input feature to the model output.

Layer-wise Relevance Propagation (LRP) [15] is a well-accepted algorithm, which is applied to compute the relevance of each input feature x_{ij} to the model outcome $\mathcal{F}_{\mathbf{x}_i}(\theta)$.

*Adaptive Laplace Mechanism: Differential Privacy Preservation in Deep Learning, Phan et.al , 2018

Algorithm 1 Adaptive Laplace Mechanism (Database D , hidden layers H , loss function $\mathcal{F}(\theta)$, and privacy budgets ϵ_1 , ϵ_2 , and ϵ_3 , the number of batches T , the batch size $|L|$)

```

1: Compute the average relevance by applying the LRP Alg.
2:  $\forall j \in [1, d] : R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i)$  #Eq.10#
3: Inject Laplace noise into the average relevance of each  $j$ -th input feature
4:  $\Delta_R = 2d/|D|$  #Lemma 1#
5: for  $j \in [1, d]$  do
6:    $\bar{R}_j \leftarrow \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) + \text{Lap}(\frac{\Delta_R}{\epsilon_1})$ 
7:  $\bar{\mathbf{R}}(D) = \{\bar{R}_j\}_{j \in [1, d]}$ 
8: Inject Laplace noise into coefficients of the differentially private layer  $h_0$ 
9:  $\Delta_{h_0} = 2 \sum_{h \in h_0} d$  #Lemma 3#
10: for  $j \in [1, d]$  do
11:    $\epsilon_j \leftarrow \beta_j \times \epsilon_2$  #Eq. 16#
12: for  $\mathbf{x}_i \in D, j \in [1, d]$  do
13:    $\bar{x}_{ij} \leftarrow x_{ij} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{h_0}}{\epsilon_j})$  #perturb input feature  $x_{ij}$ #
14:    $\bar{b} \leftarrow b + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{h_0}}{\epsilon_2})$  #perturb bias  $b$ #
15: Construct hidden layers  $\{h_1, \dots, h_k\}$  and normalization layers  $\{\bar{h}_1, \dots, \bar{h}_{(k)}\}$ 
16: Inject Laplace noise into coefficients of the approximated loss function  $\hat{\mathcal{F}}$ 
17:  $\Delta_{\mathcal{F}} = M(|\bar{h}_{(k)}| + \frac{1}{4}|\bar{h}_{(k)}|^2)$  #Lemma 5#
18: for  $\mathbf{x}_i \in D, R \in [0, 2], l \in [1, M]$  do
19:    $\bar{\phi}_{l\mathbf{x}_i}^{(R)} \leftarrow \phi_{l\mathbf{x}_i}^{(R)} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$  #perturb coefficients of  $\hat{\mathcal{F}}$ 
20: Initialize  $\theta_0$  randomly
21: for  $t \in [T]$  do
22:   Take a random training batch  $L$ 
23:   Construct differentially private affine transformation layer
24:    $\bar{\mathbf{h}}_{0L}(W_0) \leftarrow \{\bar{h}_L(W)\}_{h \in h_0}$ 
25:   s.t.  $\bar{h}_L(W) = \sum_{\mathbf{x}_i \in L} (\bar{\mathbf{x}}_i W^T + \bar{b})$ 
26:   Construct differentially private loss function
27:    $\bar{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^M \sum_{\mathbf{x}_i \in L} \sum_{R=0}^2 (\bar{\phi}_{l\mathbf{x}_i}^{(R)} W_{l(k)}^T)^R$ 
28:   Compute gradient descents
29:    $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{|L|} \nabla_{\theta_t} \bar{\mathcal{F}}_L(\theta_t)$  # $\eta_t$  is a learning rate#
30: Return  $\theta_T$  # $(\epsilon_1 + \epsilon_2 + \epsilon_3)$ -differentially private#

```

• **Step 1 (Lines 1-7).** In the first step, we obtain the average relevances of all the j -th input features, denoted as $R_j(D)$, by applying the LRP algorithm on a well-trained deep neural network on the database D . $R_j(D)$ is computed as follows:

$$R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) \quad (10)$$

Then, we derive differentially private relevances, denoted as \bar{R}_j , by injecting Laplace noise into R_j for all the j -th input features. The total privacy budget in this step is ϵ_1 .

Algorithm 1 Adaptive Laplace Mechanism (Database D , hidden layers H , loss function $\mathcal{F}(\theta)$, and privacy budgets ϵ_1 , ϵ_2 , and ϵ_3 , the number of batches T , the batch size $|L|$)

```

1: Compute the average relevance by applying the LRP Alg.
2:  $\forall j \in [1, d] : R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i)$  #Eq.10#
3: Inject Laplace noise into the average relevance of each  $j$ -th input feature
4:  $\Delta_R = 2d/|D|$  #Lemma 1#
5: for  $j \in [1, d]$  do
6:    $\bar{R}_j \leftarrow \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) + \text{Lap}(\frac{\Delta_R}{\epsilon_1})$ 
7:  $\bar{\mathbf{R}}(D) = \{\bar{R}_j\}_{j \in [1, d]}$ 
8: Inject Laplace noise into coefficients of the differentially private layer  $\mathbf{h}_0$ 
9:  $\Delta_{h_0} = 2 \sum_{h \in \mathbf{h}_0} d$  #Lemma 3#
10: for  $j \in [1, d]$  do
11:    $\epsilon_j \leftarrow \beta_j \times \epsilon_2$  #Eq. 16#
12: for  $\mathbf{x}_i \in D, j \in [1, d]$  do
13:    $\bar{x}_{ij} \leftarrow x_{ij} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{h_0}}{\epsilon_j})$  #perturb input feature  $x_{ij}$ #
14:  $\bar{b} \leftarrow b + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{h_0}}{\epsilon_2})$  #perturb bias  $b$ #
15: Construct hidden layers  $\{\mathbf{h}_1, \dots, \mathbf{h}_k\}$  and normalization layers  $\{\bar{\mathbf{h}}_1, \dots, \bar{\mathbf{h}}_{(k)}\}$ 
16: Inject Laplace noise into coefficients of the approximated loss function  $\bar{\mathcal{F}}$ 
17:  $\Delta_{\mathcal{F}} = M(|\bar{\mathbf{h}}_{(k)}| + \frac{1}{4}|\bar{\mathbf{h}}_{(k)}|^2)$  #Lemma 5#
18: for  $\mathbf{x}_i \in D, R \in [0, 2], l \in [1, M]$  do
19:    $\bar{\phi}_{l\mathbf{x}_i}^{(R)} \leftarrow \phi_{l\mathbf{x}_i}^{(R)} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$  #perturb coefficients of  $\bar{\mathcal{F}}$ 
20: Initialize  $\theta_0$  randomly
21: for  $t \in [T]$  do
22:   Take a random training batch  $L$ 
23:   Construct differentially private affine transformation layer
24:    $\bar{\mathbf{h}}_{0L}(W_0) \leftarrow \{\bar{h}_L(W)\}_{h \in \mathbf{h}_0}$ 
25:   s.t.  $\bar{h}_L(W) = \sum_{\mathbf{x}_{ij} \in L} (\bar{\mathbf{x}}_i W^T + \bar{b})$ 
26:   Construct differentially private loss function
27:    $\bar{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^M \sum_{\mathbf{x}_i \in L} \sum_{R=0}^2 (\bar{\phi}_{l\mathbf{x}_i}^{(R)} W_{l(k)}^T)^R$ 
28:   Compute gradient descents
29:    $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{|L|} \nabla_{\theta_t} \bar{\mathcal{F}}_L(\theta_t)$  # $\eta_t$  is a learning rate#
30: Return  $\theta_T$ . # $(\epsilon_1 + \epsilon_2 + \epsilon_3)$ -differentially private#

```

• **Step 2 (Lines 8-14).** In the second step, we derive a differentially private affine transformation layer, denoted \mathbf{h}_0 . Every hidden neuron $h_{0j} \in \mathbf{h}_0$ will be perturbed by injecting adaptive Laplace noise into its affine transformation to preserve differential privacy given a batch L . Based on \bar{R}_j , “more noise” is injected into features which are “less relevant” to the model output, and vice-versa. The total privacy budget used in this step is ϵ_2 . The perturbed affine transformation layer is denoted as $\bar{\mathbf{h}}_{0L}$ (Fig. 2).

For Figures:

Ref to: Adaptive Laplace Mechanism: Differential Privacy Preservation in Deep Learning, Phan et.al , 2018

Algorithm 1 Adaptive Laplace Mechanism (Database D , hidden layers H , loss function $\mathcal{F}(\theta)$, and privacy budgets ϵ_1 , ϵ_2 , and ϵ_3 , the number of batches T , the batch size $|L|$)

```

1: Compute the average relevance by applying the LRP Alg.
2:  $\forall j \in [1, d] : R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i)$  #Eq.10#
3: Inject Laplace noise into the average relevance of each  $j$ -th input feature
4:  $\Delta_R = 2d/|D|$  #Lemma 1#
5: for  $j \in [1, d]$  do
6:    $\bar{R}_j \leftarrow \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) + \text{Lap}(\frac{\Delta_R}{\epsilon_1})$ 
7:  $\bar{\mathbf{R}}(D) = \{\bar{R}_j\}_{j \in [1, d]}$ 
8: Inject Laplace noise into coefficients of the differentially private layer  $h_0$ 
9:  $\Delta_{h_0} = 2 \sum_{h \in h_0} d$  #Lemma 3#
10: for  $j \in [1, d]$  do
11:    $\epsilon_j \leftarrow \beta_j \times \epsilon_2$  #Eq. 16#
12: for  $\mathbf{x}_i \in D, j \in [1, d]$  do
13:    $\bar{x}_{ij} \leftarrow x_{ij} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{h_0}}{\epsilon_j})$  #perturb input feature  $x_{ij}$ #
14:  $\bar{b} \leftarrow b + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{h_0}}{\epsilon_2})$  #perturb bias  $b$ #
15: Construct hidden layers  $\{h_1, \dots, h_k\}$  and normalization layers  $\{\bar{h}_1, \dots, \bar{h}_{(k)}\}$ 
16: Inject Laplace noise into coefficients of the approximated loss function  $\hat{\mathcal{F}}$ 
17:  $\Delta_{\mathcal{F}} = M(|\bar{h}_{(k)}| + \frac{1}{4}|\bar{h}_{(k)}|^2)$  #Lemma 5#
18: for  $\mathbf{x}_i \in D, R \in [0, 2], l \in [1, M]$  do
19:    $\bar{\phi}_{l\mathbf{x}_i}^{(R)} \leftarrow \phi_{l\mathbf{x}_i}^{(R)} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$  #perturb coefficients of  $\hat{\mathcal{F}}$ 
20: Initialize  $\theta_0$  randomly
21: for  $t \in [T]$  do
22:   Take a random training batch  $L$ 
23:   Construct differentially private affine transformation layer
24:    $\bar{\mathbf{h}}_{0L}(W_0) \leftarrow \{\bar{h}_L(W)\}_{h \in h_0}$ 
25:   s.t.  $\bar{h}_L(W) = \sum_{\mathbf{x}_i \in L} (\bar{\mathbf{x}}_i W^T + \bar{b})$ 
26:   Construct differentially private loss function
27:    $\bar{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^M \sum_{\mathbf{x}_i \in L} \sum_{R=0}^2 (\bar{\phi}_{l\mathbf{x}_i}^{(R)} W_{l(k)}^T)^R$ 
28:   Compute gradient descents
29:    $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{|L|} \nabla_{\theta_t} \bar{\mathcal{F}}_L(\theta_t)$  # $\eta_t$  is a learning rate#
30: Return  $\theta_T$  # $(\epsilon_1 + \epsilon_2 + \epsilon_3)$ -differentially private#

```

• **Step 3 (Line 15).** In the third step, we stack hidden layers $\{h_1, \dots, h_k\}$ on top of the differentially private hidden layer \bar{h}_{0L} to construct the deep private neural network (Fig. 2). The computations of h_1, \dots, h_k are done based on the differentially private layer \bar{h}_{0L} without accessing any information from the original data. Therefore, the computations do not disclose any information. Before each stacking operation, a normalization layer, denoted \bar{h} , is applied to bound non-linear activation functions, such as ReLUs (Fig. 2).

Algorithm 1 Adaptive Laplace Mechanism (Database D , hidden layers H , loss function $\mathcal{F}(\theta)$, and privacy budgets ϵ_1 , ϵ_2 , and ϵ_3 , the number of batches T , the batch size $|L|$)

```

1: Compute the average relevance by applying the LRP Alg.
2:  $\forall j \in [1, d] : R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i)$  #Eq.10#
3: Inject Laplace noise into the average relevance of each  $j$ -th input feature
4:  $\Delta_R = 2d/|D|$  #Lemma 1#
5: for  $j \in [1, d]$  do
6:    $\bar{R}_j \leftarrow \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) + \text{Lap}(\frac{\Delta_R}{\epsilon_1})$ 
7:  $\bar{\mathbf{R}}(D) = \{\bar{R}_j\}_{j \in [1, d]}$ 
8: Inject Laplace noise into coefficients of the differentially private layer  $\mathbf{h}_0$ 
9:  $\Delta_{\mathbf{h}_0} = 2 \sum_{h \in \mathbf{h}_0} d$  #Lemma 3#
10: for  $j \in [1, d]$  do
11:    $\epsilon_j \leftarrow \beta_j \times \epsilon_2$  #Eq. 16#
12: for  $\mathbf{x}_i \in D, j \in [1, d]$  do
13:    $\bar{x}_{ij} \leftarrow x_{ij} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j})$  #perturb input feature  $x_{ij}$ #
14:  $\bar{b} \leftarrow b + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$  #perturb bias  $b$ #
15: Construct hidden layers  $\{\mathbf{h}_1, \dots, \mathbf{h}_k\}$  and normalization layers  $\{\bar{\mathbf{h}}_1, \dots, \bar{\mathbf{h}}_{(k)}\}$ 
16: Inject Laplace noise into coefficients of the approximated loss function  $\hat{\mathcal{F}}$ 
17:  $\Delta_{\mathcal{F}} = M(|\bar{\mathbf{h}}_{(k)}| + \frac{1}{4}|\bar{\mathbf{h}}_{(k)}|^2)$  #Lemma 5#
18: for  $\mathbf{x}_i \in D, R \in [0, 2], l \in [1, M]$  do
19:    $\bar{\phi}_{l|\mathbf{x}_i}^{(R)} \leftarrow \phi_{l|\mathbf{x}_i}^{(R)} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$  #perturb coefficients of  $\hat{\mathcal{F}}$ 
20: Initialize  $\theta_0$  randomly
21: for  $t \in [T]$  do
22:   Take a random training batch  $L$ 
23:   Construct differentially private affine transformation layer
24:    $\bar{\mathbf{h}}_{0L}(W_0) \leftarrow \{\bar{\mathbf{h}}_L(W)\}_{h \in \mathbf{h}_0}$ 
25:   s.t.  $\bar{\mathbf{h}}_L(W) = \sum_{\mathbf{x}_i \in L} (\bar{\mathbf{x}}_i W^T + \bar{b})$ 
26:   Construct differentially private loss function
27:    $\bar{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^M \sum_{\mathbf{x}_i \in L} \sum_{R=0}^2 (\bar{\phi}_{l|\mathbf{x}_i}^{(R)} W_{l(k)}^T)^R$ 
28:   Compute gradient descents
29:    $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{|L|} \nabla_{\theta_t} \bar{\mathcal{F}}_L(\theta_t)$  # $\eta_t$  is a learning rate#
30: Return  $\theta_T$  # $(\epsilon_1 + \epsilon_2 + \epsilon_3)$ -differentially private#

```

• **Step 4 (Lines 16-19).** After constructing a private structure of hidden layers $\{\bar{\mathbf{h}}_{0L}, \mathbf{h}_1, \dots, \mathbf{h}_k\}$, we need to protect the labels y_i at the output layer. To achieve this, we derive a polynomial approximation of the loss function \mathcal{F} . Then, we perturb the loss function \mathcal{F} by injecting Laplace noise with a privacy budget ϵ_3 into its coefficients to preserve differential privacy on each training batch L , denoted $\bar{\mathcal{F}}_L(\theta)$.

Algorithm 1 Adaptive Laplace Mechanism (Database D , hidden layers H , loss function $\mathcal{F}(\theta)$, and privacy budgets ϵ_1 , ϵ_2 , and ϵ_3 , the number of batches T , the batch size $|L|$)

```

1: Compute the average relevance by applying the LRP Alg.
2:  $\forall j \in [1, d] : R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i)$  #Eq.10#
3: Inject Laplace noise into the average relevance of each  $j$ -th input feature
4:  $\Delta_R = 2d/|D|$  #Lemma 1#
5: for  $j \in [1, d]$  do
6:    $\bar{R}_j \leftarrow \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) + \text{Lap}(\frac{\Delta_R}{\epsilon_1})$ 
7:  $\bar{\mathbf{R}}(D) = \{\bar{R}_j\}_{j \in [1, d]}$ 
8: Inject Laplace noise into coefficients of the differentially private layer  $\mathbf{h}_0$ 
9:  $\Delta_{\mathbf{h}_0} = 2 \sum_{h \in \mathbf{h}_0} d$  #Lemma 3#
10: for  $j \in [1, d]$  do
11:    $\epsilon_j \leftarrow \beta_j \times \epsilon_2$  #Eq. 16#
12: for  $\mathbf{x}_i \in D, j \in [1, d]$  do
13:    $\bar{x}_{ij} \leftarrow x_{ij} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j})$  #perturb input feature  $x_{ij}$ #
14:  $\bar{b} \leftarrow b + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$  #perturb bias  $b$ #
15: Construct hidden layers  $\{\mathbf{h}_1, \dots, \mathbf{h}_k\}$  and normalization layers  $\{\bar{\mathbf{h}}_1, \dots, \bar{\mathbf{h}}_{(k)}\}$ 
16: Inject Laplace noise into coefficients of the approximated loss function  $\hat{\mathcal{F}}$ 
17:  $\Delta_{\mathcal{F}} = M(|\bar{\mathbf{h}}_{(k)}| + \frac{1}{4}|\bar{\mathbf{h}}_{(k)}|^2)$  #Lemma 5#
18: for  $\mathbf{x}_i \in D, R \in [0, 2], l \in [1, M]$  do
19:    $\bar{\phi}_{l\mathbf{x}_i}^{(R)} \leftarrow \phi_{l\mathbf{x}_i}^{(R)} + \frac{1}{|L|} \text{Lap}(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$  #perturb coefficients of  $\hat{\mathcal{F}}$ 
20: Initialize  $\theta_0$  randomly
21: for  $t \in [T]$  do
22:   Take a random training batch  $L$ 
23:   Construct differentially private affine transformation layer
24:    $\bar{\mathbf{h}}_{0L}(W_0) \leftarrow \{\bar{h}_L(W)\}_{h \in \mathbf{h}_0}$ 
25:   s.t.  $\bar{h}_L(W) = \sum_{\mathbf{x}_{x_i} \in L} (\bar{\mathbf{x}}_i W^T + \bar{b})$ 
26:   Construct differentially private loss function
27:    $\bar{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^M \sum_{\mathbf{x}_i \in L} \sum_{R=0}^2 (\bar{\phi}_{l\mathbf{x}_i}^{(R)} W_{l(k)}^T)^R$ 
28:   Compute gradient descents
29:    $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{|L|} \nabla_{\theta_t} \bar{\mathcal{F}}_L(\theta_t)$  # $\eta_t$  is a learning rate#
30: Return  $\theta_T$  # $(\epsilon_1 + \epsilon_2 + \epsilon_3)$ -differentially private#

```

• **Step 5 (Lines 20-30).** Finally, the parameter θ_T is derived by minimizing the loss function $\bar{\mathcal{F}}_L(\theta)$ on T training steps sequentially. In each step t , stochastic gradient descent (SGD) algorithm is used to update parameters θ_t given a random batch L of training samples in D . This essentially is an optimization process, without using any additional information from the original data.

Other approaches: 2. Label Differential Privacy*

- In the case of DP-SGD, the accuracy remains significantly lower than that of models trained without DP constraints.
- Notably, for the widely considered CIFAR-10 dataset, the highest reported accuracy for DP models is 69.3%. Even using pre-training with external (CIFAR-100) data, the best reported DP accuracy, 73%², is still far below the non-private baselines (> 95%).
- The performance gap becomes a roadblocker for many real-world applications to adopt DP.
- Special case where the DP guarantee is only required to hold with respect to the labels.

Other approaches: 2. Label Differential Privacy*

In the label differential privacy (LabelDP) setting, the labels are considered sensitive, and their privacy needs to be protected, while the input points are not sensitive.

Examples include: (i) computational advertising where the impressions are known to the Ad Tech³, and thus considered non-sensitive, while the conversions reveal user interest and are thus private,

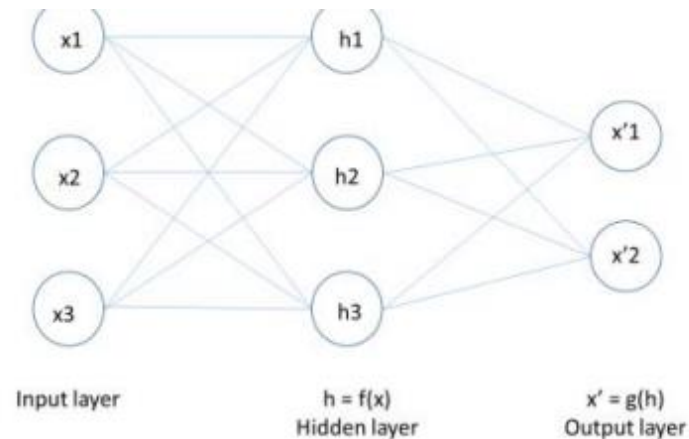
(ii) recommendation systems where the choices are known, e.g., to the streaming service provider, but the user ratings are considered sensitive, and (iii) user surveys and analytics where demographic information (e.g., age, gender) is non-sensitive but income is sensitive

Results: On CIFAR-10, we show that it achieves 20% higher accuracy than DP-SGD.

- ***Deep Learning with Label Differential Privacy, NeurIPS 2021**

Function-Level*

- The deep private auto-encoder (dPA):
 - differential private based on the functional mechanism.
 - Supervised deep learning: encodes the input values x , using a function f .
 - It then decodes the encoded values $f(x)$, using a function g , to create output values identical to the input values as depicted in figure.



Simple schema of a basic Auto-Encoder

* Differential privacy preservation for deep auto-encoders: an application of human behavior prediction, AAI 2016

Ref

https://web.cs.ucdavis.edu/~franklin/ecs289/2010/dwork_2008.pdf

Assignment Link

<https://blog.openmined.org/differential-privacy-using-pydp/>

<https://github.com/IBM/differential-privacy-library>