# Deep Learning with Differential Privacy

Chenhao Jiang
515021910104

## Contents

**Abstract**

Machine learning techniques based on neural networks are achieving remarkable results in a wide variety of domains. Often, the training of models requires large, representative datasets, which may be crowdsourced and contain sensitive information. The models should not expose private information in these datasets. However, neural networks algorithm with differentially privacy is too difficult for me to design. Fortunately, I design a linear regression algorithm with differentially privacy and study the performance of the algorithm by experiment.

## 1 Introduction

Recent progress in neural networks has led to impressive successes in a wide range of applications, including image classification, language representation and many more. These advances are enabled, in part, by the availability of large and representative datasets for training neural networks. These datasets are often crowdsourced, and may contain sensitive information. Their use requires techniques that meet the demands of the applications while offering principled and rigorous privacy guarantees.

Some work has already been implemented, in order to combine deep learning and differential privacy. I will explain the work of M. Adabi in the following sections. Then I will introduce my linear regression algorithm with differentially privacy, prove the reliability of this algorithm and study the performance of the algorithm by experiment.

## 2 Background

### 2.1 Differential Privacy

Differential privacy constitutes a strong standard for privacy guarantees for algorithms on aggregate databases. It is defined in terms of the application-specific concept of adjacent databases. For instance, when each training dataset is a set of image-label pairs; we say that two of these sets are adjacent if they differ in a single entry, that is, if one image-label pair is present in one set and absent in the other.

A randomized mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies $(\epsilon, \delta)$-differential privacy if for any two adjacent inputs $d, d' \in \mathcal{D}$ and for any subset of outputs $S \subseteq \mathcal{R}$ it hold that

$$\Pr[M(d) \in S] \leq e^{\epsilon} \Pr[M(d') \in S] + \delta$$

The original definition of $\epsilon$-differential privacy does not include the additive term $\delta$, which allows for the probability that plain $\epsilon$-differential privacy is broken with probability $\delta$ (which is preferably smaller than $1/|d|$).

Differential privacy has several properties that make it particularly useful in applications: 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.

A common paradigm for approximating a deterministic real-valued function $f : \mathcal{D} \rightarrow \mathbb{R}$ with a differentially provate mechanism is via additive noise calibrated ro f's sensitivity $S_f$, which is defined as the maximum of the absolute distance $|f(d) - f(d')|$ where $d$ and $d'$ are adjacent inputs. For instance, the Gaussian noise mechanism is defined by

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

where $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ is the normal (Gaussian) distribution with the mean 0 and standard deviation $S_f \sigma$. A single application of the Gaussian mechanism to function $f$ of sensitivity $S_f$ satisfies $(\epsilon, \delta)$-differential privacy if $\delta \geq \frac{4}{5} exp(-(\sigma\epsilon)^2/2)$ and $\epsilon < 1$.

The basic blueprint for designing a differentially private additive-noise mechanism that implements a given functionality consists of the following steps: approximating the functionality by a sequential composition of bounded-sensitivity functions; choosing parameters of additive noise; and performing privacy analysis of the resulting mechanism.

## 2.2 Deep Learning

Deep neural networks, which are remarkably effective for many machine learning tasks, define parameterized functions from inputs to outputs as compositions of many layers of basic building blocks, such as affine transformations and simple nonlinear functions. Commonly used examples of the latter are sigmoids and rectified linear units (ReLUs). By varying parameters of these blocks, we can "train" such a parameterized function with the goal of fitting any given finite set of input/output examples.

More precisely, we define a loss function $\mathcal{L}$ that represents the penalty for mismatching the training data. The loss $\mathcal{L}(\theta)$ on parameters $\theta$ is the average of the loss over the training examples $\{x_1, \ldots x_N\}$, so $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Training consists in finding $\theta$ that yields an acceptably small loss, hopefully the smallest loss (though in practice we seldom expect to reach an global minimum).

For complex networks, the loss function $\mathcal{L}$ is usually nonconvex and difficult to minimize. In practice, the minimization is often done by the mini-batch stochastic gradient descent (SGD) algorithm. In this algorithm, at each step, one forms a batch B of random examples and computes $\mathfrak{g}_B = 1/|B| \sum_{x \in B} \nabla_\theta \mathcal{L}(\theta, x)$ as an estimation to the gradient $\nabla_\theta \mathcal{L}(\theta)$. Then $\theta$ is updated following the gradient direction $-\mathfrak{g}_B$ towards a local minimum.

Several systems have been built to support the definition of neural networks, to enable efficient training, and then to perform efficient inference. And TensorFlow, an open-source dataflow engine released by Google, allows the programmer to define large computation graphs from basic operators, and to distribute their execution across a heterogeneous distributed system. TensorFlow automates the creation of the computation graphs for gradients; it also makes it easy to batch computation.

# 3 Related Work

In this section, I will introduce **Martin Abadi** and his partner's approach toward differential private training of netural networks from the following three parts: a differentially private SGD algorithm, the moments accountant, and hyperparameter tuning.

## 3.1 Differentially Private SGD Algorithm

Algorithm 1 outlines their basic method for training a model with parameters $\theta$ by minimizing the empirical loss function $\mathcal{L}(\theta)$. At each step of the SGD, they compute the gradient $\nabla_\theta \mathcal{L}(\theta, x_i)$ for a random subset of examples, clip the $\ell_2$ norm of each gradient, compute the average, add noise in order to protect privacy, and take a step in the opposite direction of this average noisy gradient. At the end, in addition to outputting the model, it is also needed to compute the privacy loss of the mechanism based on the information maintained by the privacy accountant. Next I will describe in more detail each component of this algorithm and our refinements.

**Algorithm 1** Differentially private SGD (Outline)

```
Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate $\eta_t$, noise scale $\sigma$, group size $L$, gradient norm bound $C$.
  Initialize $\theta_0$ 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
    $\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$
    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 $\theta_T$ and compute the overall privacy cost $(\varepsilon, \delta)$ using a privacy accounting method.
```

**Norm clipping:** Proving the differential privacy guarantee of Algorithm 1 requires bounding the influence of each individual example on $\tilde{\mathfrak{g}}_t$. Since there is

no a priori bound on the size of the gradients, they clip each gradient in $\ell_2$ norm; i.e., the gradient vector $\mathfrak{g}$ is replaced by $\mathfrak{g}/max(1, \frac{\|\mathfrak{g}\|_2}{C})$, for a clipping threshold $C$. This clipping ensures that if $\|\mathfrak{g}\|_2 \leq C$, then $\mathfrak{g}$ is preserved, whereas if $\|\mathfrak{g}\|_2 > C$, it gets scaled down to be of norm $C$. They remark that gradient clipping of this norm is a popular ingredient of SGD for deep networks for non-privacy reasons, though in that setting it usually suffices to clip after averaging.
**Per-layer and time-dependent parameters:** The pseudocode for Algorithm 1 groups all the parameters into a single input $\theta$ of the loss function $\mathcal{L}(\cdot)$. For multi-layer neural networks, they consider each layer separately, which allows setting different clipping thresholds $C$ and noise scales $\sigma$ for different layers. Additionally, the clipping and noise parameters may vary with the number of training steps $t$.
**Lots:** Like the ordinary SGD algorithm, Algorithm 1 estimates the gradient of $\mathcal{L}$ by computing the gradient of the loss on a group of examples and taking the average. This average provides an unbiased estimator, the variance of which decreases quickly with the size of the group. We call such a group a lot, to distinguish it from the computational grouping that is commonly called a batch. In order to limit memory consumption, we may set the batch size much smaller than the lot size $L$, which is a parameter of the algorithm. We perform the computation in batches, then group several batches into a lot for adding noise. In practice, for efficiency, the construction of batches and lots is done by randomly per- muting the examples and then partitioning them into groups of the appropriate sizes. For ease of analysis, however, we assume that each lot is formed by independently picking each example with probability $q = L/N$ , where $N$ is the size of the input dataset.

As is common in the literature, we normalize the running time of a training algorithm by expressing it as the number of epochs, where each epoch is the (expected) number of batches required to process $N$ examples. In our notation, an epoch consists of $N/L$ lots.
**Privacy accounting:** For differentially private SGD, an important issue is computing the overall privacy cost of the training. The composability of differential privacy allows them to implement an "accountant" procedure that compute the privacy cost at each access to the training data, and accumulates this cost as the training progresses. Each step of training typically requires

gradients at multiple layers, and the accountant accumulates the cost that corresponds to all of them.
**Moments accountant:** Much research has been devoted to studying the privacy loss for a particular noise distribution as well as the composition of privacy losses. For the Gaussian noise that they use, if they choose $\sigma$ in Algorithm 1 to be $\sqrt{2\log\frac{1.25}{\delta}}/\epsilon$, then by standard arguments each step is $(\epsilon, \delta)$-differentially private with respect to the lot. Since the lot itself is a random sample from the database, the privacy amplification theorem implies that each step is $(q\epsilon, q\delta)$-differentially private with respect to the full database where $q = L/N$ is the sampling ratio per lot. The result in the literature that yields the best overall bound is the strong composition theorem.

However, the strong composition theorem can be loose, and does not take into account the particular noise distribution under consideration. In their work, they invent a stronger accounting method, which they call the moments accountant. It allows them to prove that Algorithm 1 us $O(q\epsilon\sqrt{T}), \delta)$-differentially private for appropriately chosen settings of the noise scale and the clipping threshold. Compared to what one would obtain by the strong composition theorem, their bound is tighter in two ways: it saves a $\sqrt{\log(1/\delta)}$ factor in the $\epsilon$ part and a $Tq$ factor in the $\delta$ part. Since they expect $\delta$ to be small and $T \gg 1/q$ (i.e.,each example is examined multiple times), the saving provided by our bound is quite significant. This result is one of their contributions.

**Theorem 3.1.1.** *There exist constants $c_1$ and $c_2$ so that given the sampling probability $q = L/N$ and the number of steps $T$, for any $\epsilon < c_1 q^2 T$, Algorithm 1 is $(\epsilon, \delta)$-differentially private for any $\delta > 0$ if we choose*

$$\sigma \geq c_2 \frac{q\sqrt{T\log(1/\delta)}}{\epsilon}$$

They use the strong composition theorem, and then they need to choose $\sigma = \Omega(q\sqrt{T\log(1/\delta)\log(T/\delta)}/\epsilon)$. Note that they save a factor of $\sqrt{\log(T/\delta)}$ in their asymptotic bound. The moments accountant is beneficial in theory, as this result indicates, and also in practice. For example, with $L = 0.01N$, $\sigma = 4$, $\delta = 10^{-5}$, and $T = 10000$, we have $\epsilon \approx 1.26$ using the moments accountant. As a comparison, we would get a much larger $\epsilon \approx 9.34$ using the strong composition theorem.

## 3.2 The Moments Accoutant

The moments accountant keeps track of a bound on the moments of the privacy loss random variable. It generalizes the standard approach of tracking $(\epsilon, \delta)$ and using the strong composition theorem. While such an improvement was known previously for composing Gaussian mechanisms, they show that it applies also for composing Gaussian mechanisms with random sampling and can provide much tighter estimate of the privacy loss of Algorithm 1.

Privacy loss is a random variable dependent on the random noise added to the algorithm. That a mechanism $\mathcal{M}$ is $(\epsilon, \delta)$-differentially private is equivalent to a certain tail bound on $\mathcal{M}$' s privacy loss random variable. While the tail bound is very useful information on a distribution, composing directly from it can result in quite loose bounds. They instead compute the log moments of the privacy loss random variable, which compose linearly. They then use the moments bound, together with the standard Markov inequality, to obtain the tail bound, that is the privacy loss in the sense of differential privacy.

More specifically, for neighboring databases $d, d' \in \mathcal{D}^n$, a mechanism $\mathcal{M}$, auxiliary input $aux$, and an outcome $o \in \mathcal{R}$ , define the privacy loss at $o$ as

$$c(o; \mathcal{M}, aux, d, d') \triangleq log \frac{\Pr[\mathcal{M}(aux, d) = o]}{Pr[\mathcal{M}(aux, d') = o]}$$

A common design pattern, which they use extensively in the paper, is to update the state by sequentially applying differentially private mechanisms. This is an instance of adaptive composition, which they model by letting the auxiliary input of the $k^{th}$ mechanism $\mathcal{M}_k$ be the output of all the previous mechanisms.

For a given mechanism $\mathcal{M}$, they define the $\lambda^{th}$ moment $\alpha_{\mathcal{M}}(\lambda; aux, d, d')$ as the log of the moment generating function evaluated at the value $\lambda$:

$$\alpha_{\mathcal{M}}(\lambda; aux, d, d') \triangleq \log \mathbb{E}_{o \backsim \mathcal{M}(aux,d)}[exp(\lambda c(o; \mathcal{M}, aux, d, d'))].$$

In order to prove privacy guarantees of a mechanism, it is useful to bound all possible $\alpha_{\mathcal{M}}(\lambda; aux, d, d')$. We define

$$\alpha_{\mathcal{M}}(\lambda) \triangleq \max_{aux,d,d'} \alpha_{\mathcal{M}}(\lambda; aux, d, d'),$$

where the maximum is taken over all possible $aux$ and all the neighboring databases $d, d'$.

They state the properties of $\alpha$ that we use for the moments accountant.

**Theorem 3.2.1.** *Let $\alpha_{\mathcal{M}}(\lambda)$ defined as above. Then*
*1.[**Composability**] Suppose that a mechanism $\mathcal{M}$ consists of a sequence of adaptive mechanisms mathcalM$_1, \ldots, \mathcal{M}_k$ where $\mathcal{M}_i : \prod_{j=1}^{i-1} \mathcal{R}_j \times \mathcal{D} \to \mathcal{R}_i$. Then for any $\lambda$*

$$\alpha_{\mathcal{M}}(\lambda) \leq \sum_{i=1}^{k} \alpha_{\mathcal{M}_i}(\lambda).$$

*2.[**Tail bound**] For any $\epsilon > 0$, the mechanism $\mathcal{M}$ os $(\epsilon, \delta)$-differentially private for*

$$\delta = \min_{\lambda} exp(\alpha_{\mathcal{M}}(\lambda) - \lambda\epsilon).$$

In particular, Theorem 3.2.1.1 holds when the mechanisms themselves are chosen based on the (public) output of the previous mechanisms.

By Theorem 3.2.1, it suffices to compute, or bound, $\alpha_{\mathcal{M}_i}(\lambda)$ at each step and sum them to bound the moments of the mechanism overall. They then use the tail bound to convert the moments bound to the $(\epsilon, \delta)$-differential privacy guarantee.

The main challenge that remains is to bound the value $\alpha_{\mathcal{M}_t}(\lambda)$ for each step. In the case of a Gaussian mechanism with random sampling, it suffices to estimate the following moments. Let $\mu_0$ denote the probability density function of $\mathcal{N}(0, \sigma^2)$, and $\mu_1$ denote the pdf of $\mathcal{N}(1, \sigma^2)$. Let $\mu$ be the mixture of two Gaussian $\mu = (1-q)\mu_0 + q\mu_1$. Then we need to compute $\alpha(\lambda) = \log\max(E_1, E_2)$ where

$$E_1 = \mathbb{E}_{z \backsim \mu_0}[(\mu_0(z)/\mu(z))^{\lambda}],$$
$$E_2 = \mathbb{E}_{z \backsim \mu}[(\mu(z)/\mu_0(z))^{\lambda}],$$

In the implemention of the moments accoutant, they carry out numerical integration to compute $\alpha(\lambda)$. In addition, they show the asymptotic bound

$$\alpha(\lambda) \leq q^2\lambda(\lambda+1)/(1-q)\sigma^2 + O(q^3/\sigma^3)$$

Together with Theorem 3.2.1, the above bound implies our main Theorem 3.1.1.

### 3.3 Hyperparameter Tuning

They identify characteristics of models relevant for privacy and, specifically, hyperparameters that they tune in order to balance privacy, accuracy, and performance. In particular, through experiments, they observe that model accuracy is more sensitive to training parameters such as batch size and noise level than to the structure of a neural network.

After they try several settings for the hyperparameters, they trivially add up the privacy costs of all the settings, possibly via the moments accountant. However, since they care only about the setting that gives us the most accurate model, they can do better. They use insights from theory to reduce the number of hyperparameter settings that need to be tried. While differentially private optimization of convex objective functions is best achieved using batch sizes as small as 1, non-convex learning, which is inherently less stable, benefits from aggregation into larger batches. At the same time, Theorem 1 suggests that making batches too large increases the privacy cost, and a reasonable tradeoff is to take the number of batches per epoch to be of the same order as the desired number of epochs. The learning rate in non-private train- ing is commonly adjusted downwards carefully as the model converges to a local optimum. In contrast, they never need to decrease the learning rate to a very small value, because differentially private training never reaches a regime where it would be justified. On the other hand, in their experiments, they do find that there is a small benefit to starting with a relatively large learning rate, then linearly decaying it to a smaller value in a few epochs, and keeping it constant afterwards.

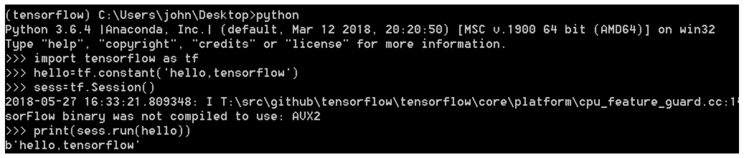
## 4 My Work

### 4.1 Tensorflow Installation

TensorFlow is Google's second-generation artificial intelligence learning system based on DistBelief's R&D. Its name comes from its own operating principle. Tensor (Tension) means an N-dimensional array, Flow (Flow) implies calculation based on a data flow graph, TensorFlow flows from one end of the flow graph to the other. TensorFlow is a system that transmits complex data structures to artificial intelligence neural networks for analysis and processing.

TensorFlow supports CNN, RNN, and LSTM algorithms, which are the most popular deep neural network models currently used in Image, Speech, and NLP. Therefore, TensorFlow can be used in many machine learning and deep learning areas such as speech recognition or image recognition.

According to the guidance of a CSDN blog on the Internet, I completed the installation of TensorFlow to facilitate the testing of code. And I learned some of the TensorFlow programming methods so that I can understand the predecessors' codes. As we can see from the following figure, after testing, we found that, we found that Tensorflow can be successfully applied.



```
(tensorflow) C:\Users\john\Desktop>python
Python 3.6.4 |Anaconda, Inc.| (default, Mar 12 2018, 20:20:50) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> hello=tf.constant('hello,tensorflow')
>>> sess=tf.Session()
2018-05-27 16:33:21.809348: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:1
sorFlow binary was not compiled to use: AVX2
>>> print(sess.run(hello))
b'hello,tensorflow'
```

Figure 1: Tensorflow can be applied

## 4.2 Linear Regression with Differentially Privacy

I try to write a differentially private SGD algorithm as M. Abadi, and I try to write a deep learning algorithm with differential privacy. However, I failed. At no choice, I decide to write a linear regression algorithm with differentially privacy. What I do is writing a linear regression algorithm with differentially privacy and studying the performance of this algorithm, compared to the linear regression algorithm without differentially privacy.

### 4.2.1 Algorithm Design

We all know what is linear regression. In linear regression, we define loss function as

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2,$$

where $m$ is number of training sets, $x^{(i)} = (x_0^{(i)}, x_1^{(i)}, \dots, x_d^{(i)})^T$ is features of $i^{th}$ training example or "input variable" of $i^{th}$ training example (there are $d+1$ dimension features, $x_0 = 1$), $y^{(i)}$ is target variable of $i^{th}$ training example, $h_\theta(x)$ is Hypothesis and $h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$. What we want to do is to determine $\theta = (\theta_0, \dots, \theta_d)^T$ which minimizes the loss function. We often use gradient descent algorithm to determine $\theta$.

We can use feature scale to process the data. We can make every $x_k^{(i)}$ and $y^{(i)}$ range from [-1,1].

However, directly posting $\theta$ will reveal private information and violate $\epsilon$-differential privacy. I first make the following derivation of loss funtion

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{2m} [\sum_{0 \le j,k \le d} (\sum_{i=1}^{m} x_j^{(i)} x_k^{(i)}) \theta_j \theta_k - 2 \sum_{j=0}^{d} (\sum_{i=1}^{m} y^{(i)} x_j^{(i)}) \theta_j + \sum_{i=1}^{m} {y^{(i)}}^2] \end{aligned}$$

We can regard loss function as $\theta$'s quadratic function, it can be simplified to

$$J(\theta) = a\theta^2 + b\theta + c$$

I split the loss function into two child functions $g(\theta)$ and $t(\theta)$, where $g(\theta) = a\theta^2$, $t(\theta) = b\theta$. Let $\Delta_1$ indicates the sensitivity of $g(\theta)$, $\Delta_2$ indicates the sensitivity of $b(\theta)$. As for a quadratic function, the loss function has a value of 0 ideally, the optimal $\theta$ is $-\frac{b}{2a}$, not related to $c$. Therefore, the algorithm can ignore $c$ when solving optimal model parameters $\bar{\theta}$. Assume that two adjacent dataset are different in last training example. We can calculate $\Delta_1, \Delta_2$ according to the definition of sensitivity $\Delta_f = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|$:

$$\Delta_1 = \sum_{0 \le j,k \le d} \|x_j^{(n)} x_k^{(n)} - x_j^{'(n)} x_k^{'(n)}\| \le 2(d+1)^2$$

$$\Delta_2 = \sum_{j=0}^{d} \|(-2y^{(n)} x_j^{(n)} - (-2y^{'(n)} x_j^{'(n)}\| \le 4(d+1)$$

According to function $g(\theta)$ and $t(\theta)$, we assign privacy budget $\epsilon_1, \epsilon_2$. According to the characteristics of the Laplace mechanism, the Laplacian noise is proportional to the sensitivity of the function and inversely proportional to the privacy budget. In order to reduce the added noise, we allocate a larger privacy budget for the more sensitive functions. According to the computed $\Delta_1, \Delta_2$, when $d$ is large, the sensitivity of $g(\theta)$ is larger than $t(\theta)$. Thus, we should make $\epsilon_1 \geq \epsilon_2$, i.e., for a fixed privacy budget $\epsilon$, we can reduce noise by rationally allocating $\epsilon_1$ and $\epsilon_2$. And it is more accurate to predict linear regression models.

In general, I want to summarize my algorithm. First, I assign privacy budget $\epsilon_1$ and $\epsilon_2$. And then I split the loss function into two child functions $g(\theta)$ and $t(\theta)$, where $g(\theta) = a\theta^2$, $t(\theta) = b\theta$. After doing this, I add Laplacian noise to the coefficient $a$ and $b$ as $\hat{a} = a + Lap(\frac{\Delta_1}{\epsilon_1})$, $\hat{b} = b + Lap(\frac{\Delta_2}{\epsilon_2})$. And then optimize the new loss function to get $\hat{\theta}$.

### 4.2.2 Algorithms Analysis

I first want to prove that function $\hat{g}(\theta)$ satisfies $\epsilon_1$-differentially privacy, where $\hat{g}(\theta) = \hat{a}\theta^2$.

**Proof:** Given two adjacent datasets $D_1$, $D_2$ different in last training example. We have

$$\begin{aligned}
\frac{\Pr\{\hat{g}(\theta)|D_1\}}{\Pr\{\hat{g}(\theta)|D_2\}} &= \frac{\prod_{0\leq j,k\leq d} exp(\frac{\epsilon_1 \| \sum_{D_1}(x_j^{(i)}x_k^{(i)} - \hat{a}\|}{\Delta_1})}{\prod_{0\leq j,k\leq d} exp(\frac{\epsilon_1 \| \sum_{D_2}(x_j^{(i)}x_k^{(i)} - \hat{a}\|}{\Delta_1})} \\
&\leq \prod_{0\leq j,k\leq d} exp(\frac{\epsilon_1}{\Delta_1}\|(\sum_{D_1} x_j^{(i)}x_k^{(i)}) - (\sum_{D_2} x_j^{(i)}x_k^{(i)})\|) \\
&= \prod_{0\leq j,k\leq d} exp(\frac{\epsilon_1}{\Delta_1}\|x_j^{(n)}x_k^{(n)} - x_j^{'(n)}x_k^{'(n)}\|) \\
&= exp(\frac{\epsilon_1}{\Delta_1}\sum_{0\leq j,k\leq d}\|x_j^{(n)}x_k^{(n)} - x_j^{'(n)}x_k^{'(n)}\|) \\
&\leq exp(\epsilon_1)
\end{aligned}$$

Also, we can prove that $\hat{t}(\theta)$ satisfies $\epsilon_2$-differentially privacy. Therefore, the algorithm satisfies $\epsilon$-differentially privacy, where $\epsilon = \epsilon_1 + \epsilon_2$.

#### 4.2.3 Algorithm Performance Test

We use Matlab to test the algorithm, the dataset is US and Brazil from Integrated Public Use Microdata, including 370000 examples and 190000. We first use feature scale to process the data. And then we randomly take 80% examples as training examples and 20% as test examples.

We perform experiments on datasets US and Brazil, respectively. For different datasets and different privacy budget values $\epsilon$, the experimental results of our linear regression algorithm with differentially privacy, compared to linear regression algorithm, are shown in the following figure.

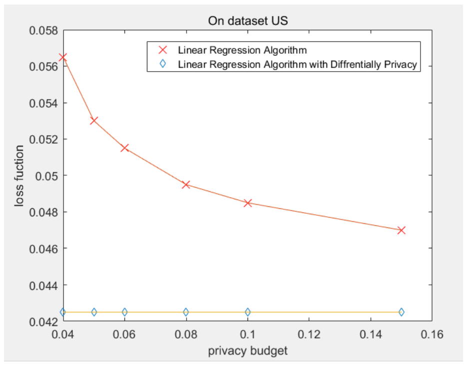


Figure 2: Algorithm performance on dataset US

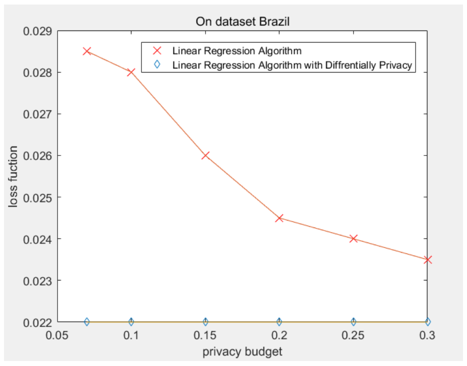


Figure 3: Algorithm performance on dataset Brazil

As we can see, the loss function of linear regression algorithm with differentially privacy is larger than the loss function of linear regression algorithm on both datasets. And the loss function of linear regression algorithm with differentially privacy reduces with the increase of privacy budget.

## References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I.Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. CoRR, abs/1607.00133, 2016.

[2] CSDN Blog
https://blog.csdn.net/cs_hnu_scw/article/details/79695347

[3] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] C. Dwork and J. Lei. Differential privacy and robust statistics. In STOC, pages 371 – 380. ACM, 2009.

[5] C. Dwork and G. N. Rothblum. Concentrated differential privacy. CoRR, abs/1603.01887, 2016.

[6] Minnesota Population Center. Ingrated public use micodata series-internation: Version 5.0[OL].[2009]. https://internationnal.ipums.org.