Report project of fairness and privacy in machine learning: The Discrete Gaussian for Differential Privacy

Mohamed Fyras Telmini, Luca Teodorescu

January 28, 2024

Abstract

This report serves to explain the different results explicited by [1]. In the first part we introduce the notions of privacy used in the article, notably the notion of **Concentrated differential privacy**. The second part concerns the results of [2] which we felt are important to explain, as they justify the necessity of an algorithm that samples from a discrete gaussian instead of using one that samples from a continuous gaussian. The third part serves to show the main results of [1] in terms of the privacy, utility and sampling of the discrete gaussian distribution. The fourth part lists the various algorithms used to obtain these samples, each followed by its corresponding python code snippet that we obtained from [1] which is the repository that the researchers provided. The last part serves to show the main advantage that the discrete gaussian has over the more commonly used discrete Laplace, which is the ease of expression of the privacy criterions in the context of the composition of queries.

1 Notions of privacy

In this article, the privacy definition we saw in our course is introduced as **Pure/Approximate Differential privacy**. The definition for which is as follows:

Definition 2 (Pure/Approximate Differential Privacy):

A randomized algorithm $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ satisfies (ϵ, δ)-differential privacy if, for all $x, x' \in \mathcal{X}^n$ that differ on a single element, and all events $E \subseteq \mathcal{Y}$, we have :

$$P[\mathcal{M}(x) \in E] \leq e^{\epsilon} P[\mathcal{M}(x') \in E] + \delta.$$

The special case of $(\epsilon, 0)$-differential privacy is referred to as pure or pointwise ϵ-differential

privacy. Whereas, for $\delta > 0$, (ϵ, δ)-differential privacy is referred to as approximate differential privacy.

In other words, approximate differential privacy allows for a small probability δ that the privacy guarantee of the mechanism is not satisfied.

In addition to it, another privacy notion was introduced that wasn't in our course named **Concentrated Differential Privacy**. This notion of privacy uses the Renyi divergence to express differential privacy. The Renyi divergence of order α between P and Q is defined as:

$$D_\alpha(P||Q) = \frac{1}{\alpha - 1} \log \sum_{y \in \mathcal{Y}} P(y)^\alpha Q(y)^{1-\alpha}$$

where P and Q are the two probability distributions, α is the Renyi divergence parameter, $\mathcal{Y}$ is the set of all possible outcomes.

Concentrated Differential Privacy is then defined as:

Definition 3 (Concentrated Differential Privacy): A randomized algorithm $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ satisfies $\frac{\epsilon^2}{2}$-concentrated differential privacy if, for all $x, x' \in \mathcal{X}^n$ that differ on a single element, and for all $\alpha \in (1, \infty)$, we have $D_\alpha(\mathcal{M}(x)||\mathcal{M}(x')) \leq \frac{\epsilon^2}{2}\alpha$

Where D_α is the Renyi divergence of order α, and ϵ is the privacy parameter

The paper also introduces a corollary that allows the conversion from Pure/Approximate Differential privacy to Concentrated Differential Privacy. We chose, however, to not introduce it, as we couldn't understand how it's applied.

2 Vulnerability of continuous noise

To understand correctly the context in which Discrete Gaussian sampling is used we have to have in mind the current limits that a naive implementation of Gaussian sampling may induce. This issue is clearly presented by [2]. To better understand this contextualization we have to bare in mind that such limits occur only because of "material" limitation and processing design. The first concept to grasp is how floating numbers are considered in programming languages and in particular double-precision floating numbers, called doubles. Such doubles are traditionally defined in 64 bits ; 1 bit for sign, 11 bits for exponent and 52 bits for the significand. Let s be the value of the sign (0 for positive and 1 for negative), $(d_1, d_2, \ldots, d_{52})$ the values for the significand and e the exponent. Thus we can write any floating number in the form of :

$$(-1)^s \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}.$$

This is the general idea of the way floating point numbers are written in programming languages. There are some details such as the value 0 for e is reserved for special number such as

NaN.

The main take-away from such a construction is that these numbers are non-uniform in the real set. This issue is one important aspect of the work designing algorithmic random simulators in floating range.

The issue of a random number generator comes when building some more mathematical distribution functions because of the specificity of double-precision floating numbers. So morally if we apply certain functions to have a distribution some numbers of the set of doubles will be sent on already existing numbers of this set because you cannot get a floating that is not in the set (as it cannot be defined). So some doubles will artificially appear more than once and some will not appear thus altering the distribution in the end.

The idea of an attack that is using this problem consists into checking if certain doubles are in the support of the distributions and seeing that an output is not feasible is an evidence that can rule out certain input between the possible ones.

3 Privacy of the discrete gaussian

Definition 1 (Discrete Gaussian). Let μ, $\sigma \in R$ with $\sigma > 0$. The discrete Gaussian distribution with location μ and scale σ is denoted $N_Z(\mu, \sigma^2)$. It is a probability distribution supported on the integers and defined by:

$$\forall x \in Z, P_{X \leftarrow N_Z(\mu,\sigma^2)}[X = x] = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\Sigma_{y \in R} e^{-\frac{(y-\mu)^2}{2\sigma^2}}}$$

In the article, there were demonstrations of the various properties satisfied by the discrete gaussian. These properties are:

3.1 Concentrated Differential Privacy

The paper stated that one of the main results the authors proved was that the discrete gaussian satisfies concentrated differential privacy. Accordingly, we feel that the proof for this is worth going through, but we will do so by providing clear explanations with minimal mathematical notations.

The definition of $\frac{1}{2}\epsilon^2$-concentrated differential privacy states that for a randomized algorithm M that gives an output Y from a dataset X (for example, a sum of the values of a column from the input dataset), the Renyi divergence of order α from two instances of the input dataset that differ on a single element is bounded by $\frac{1}{2}\epsilon^2\alpha$. In our case, the randomized algorithm M takes the following form: M(x) = q(x)+Y. q(x) represents the true result of the algorithm without the addition of noise, and Y is the additive noise that protects the privacy of the original data (this is the main idea of differential privacy in general).

In order to obtain the $\frac{1}{2}\epsilon^2\alpha$ bound, a more general inequality was proved: the Renyi divergence of order α of two discrete gaussian distributions that have different mean values is bounded by $\frac{d^2}{2\sigma^2}.\alpha$ where d is the difference between the mean values of these distributions. The proof of this is done by simply writing the definition of the Renyi divergence of order α and inserting the definition of the discrete gaussian distribution in the formula. A trick to simplify the calculations in the exponentials present in the discrete gaussian distributions is to take the case where the first distribution has a mean value μ and the second has a mean value 0. This leads to the following equality:

$e^{(\alpha-1)D_\alpha(N_Z(\mu,\sigma^2)||N_Z(0,\sigma^2))} = \sum_{x\in Z} P_{X\leftarrow N_Z(\mu,\sigma^2)}[X=x]^\alpha P_{X\leftarrow N_Z(0,\sigma^2)}[X=x]^{1-\alpha}$

$= e^{\alpha\frac{\alpha-1}{2\sigma^2}\mu^2} \frac{\sum_{x\in Z} e^{-\frac{(x-\alpha\mu)^2}{2\sigma^2}}}{\sum_{y\in Z} e^{-\frac{(y)^2}{2\sigma^2}}}$

We note that using the exponential of the Renyi divengence just gets rid of the log term in it to further simplify the calculations. The equality is then bounded by the term $e^{\alpha\frac{\alpha-1}{2\sigma^2}\mu^2}$. This is due to the fact that $\frac{\sum_{x\in Z} e^{-\frac{(x-\alpha\mu)^2}{2\sigma^2}}}{\sum_{y\in Z} e^{-\frac{(y)^2}{2\sigma^2}}} \leq 1$. The proof of this goes as follows:

1. We concider the function f and its fourrier transform $\hat{f}(y)$ defined as:
   Let $f : R \to R$ be defined by $f(x) = e^{-\frac{x^2}{2\sigma^2}}$. Its Fourier transform is $\hat{f} : R \to R$ by $\hat{f}(y) = \int_R f(x)e^{-2\pi ixy}dx = \sqrt{2\pi\sigma^2}e^{-2\pi^2\sigma^2y^2}$.

2. The poisson summation formula gives that:
   for every $t \in R$, we have $\sum_{x\in Z} f(x+t) = \sum_{y\in Z} \hat{f}(y)e^{2\pi iyt}$.

3. By simply applying the triangle inequality to f(x-μ) we obtain $\sum_{x\in Z} f(x-\mu) \leq \sum_{y\in Z} \hat{f}(y)$ which when combined with the two previous results gives that $\sum_{x\in Z} e^{-\frac{(x-\alpha\mu)^2}{2\sigma^2}} \leq \sum_{y\in Z} e^{-\frac{(y)^2}{2\sigma^2}}$ which is equivalent to saying $\frac{\sum_{x\in Z} e^{-\frac{(x-\alpha\mu)^2}{2\sigma^2}}}{\sum_{y\in Z} e^{-\frac{(y)^2}{2\sigma^2}}} \leq 1$.

We also note that the even though the last inequality uses x on one side and y on the other, these are just indexes in the sum.

The combinations of these different results gives us $D_\alpha(N_Z(\mu,\sigma^2)||N_Z(\nu,\sigma^2)) \leq \frac{(\mu-\nu)^2}{2\sigma^2}\alpha$. And by cleverly bounding q(x) and chosing a Y that follows a fitting distribution: $|q(x)-q(x_0)| \leq \Delta$ and $Y \leftarrow N_Z(0, \frac{\Delta^2}{\varepsilon^2})$ we directly obtain the concentrated differential privacy definition.

The authors mention that this bound is also satisfied by the continuous gaussian, proving that in this case these two distributions are equally effective.

3.2 Approximate differential privacy

Similarely, the proof of the approximate differential privacy garantee of the discrete gaussian is mentionned. This proof seemed more complicated due to the introduction of an intermediate

notion called the privacy loss random variable. We chose not to go thoroughly through the proof of this because its spans four pages in the original paper, but the general idea of it is as follows:

1. First, the privacy loss random variable is introduced:$Z = f(y) = \log\left(\frac{P[M(x)=y]}{P[M(x_0)=y]}\right)$ and we write $Z \leftarrow PrivLoss(M(x), M(x_0))$
2. This new notion is used to reformulate the concentrated differential privacy $D_\alpha(M(x)||M(x_0)) = \frac{1}{\alpha-1}\log\left(E_{Z\leftarrow PrivLoss(M(x),M(x_0))}\left[e^{(\alpha-1)Z}\right]\right)$
3. it is then used to reformulate the approximate differential privacy theorem
4. The next goal was to prove the new formulation of this theorem by determining the distribution of the privacy loss random variable for the case of the discrete gaussian, this seemed to be the most complicated step, we will not explain it as we haven't fully understood it.

3.3 Composition for concentrated differential privacy

We will state this property as its shown in the paper, its mostly used to compare the discrete gaussian to the discrete laplace (which we will explicit in the last part of the report). It mainly shows that its more practical to work with discrete gaussian noise in the case of algorithms that act through composition (the result of the algorithm is fed to it) as it is easier to control the privacy criterions. The property is stated as follows:
Lemma 31 (Composition for Concentrated Differential Privacy): Let $M_1 : X^n \to Y_1$ satisfy $\frac{1}{2}\epsilon_1^2$-concentrated differential privacy. Let $M_2 : X^n \times Y_1 \to Y_2$ be such that, for all $y \in Y_1$, the restriction $M_2(., y) : X^n \to Y_2$ satisfies $\frac{1}{2}\epsilon_2^2$-concentrated differential privacy.
Define $M^* : X^n \to Y_2$ by $M^*(x) = M_2(x, M_1(x))$. Then M^* satisfies $\frac{1}{2}(\epsilon_1^2 + \epsilon_2^2)$-concentrated differential privacy.
This means that in the case of k-counting queries (counting the number of occurrences of a variable's value in a dataset, repeated for k different values), it's possible to guarantee $\frac{1}{2}(\epsilon^2)$-concentrated differential privacy by adding noise sampled from $N_Z(0, \frac{k}{\epsilon^2})$ to each singular value.

3.4 Utility

The article provides a comparison of the utility of the discrete gaussian and the continuous gaussian, they also study the utility of the rounded gaussian which is often used as a proxy for the discrete gaussian. This mostly shows the inefficiency of the rounded gaussian as a proxy, as its utility is always strictly less than that of the discrete gaussian. The discrete Gaussian has subgaussian tails.

3.5 Sampling

The paper presents an efficiant way to sample from a discrete gaussian, by using properties of bernouilli and discrete laplace distributions. The reason behind this is the following property: For a random variable Y following a Laplace distribution with parameter $\frac{1}{\tau}$ and C following a binomial distribution with parameter $\frac{1-e^{-\tau}}{1+e^{-\tau}} e^{-\frac{\sigma^2\tau^2}{2}} \sum_{y\in Z} e^{-\frac{y^2}{2\sigma^2}}$. We have that Y conditioned to C=1 follows a discrete gaussian distribution.

The proof of this property is a result of rewriting the probability $P(Y = y|C = 1)$ as that of a random variable that follows a discrete gaussian distribution, and is as such:

$$P[X = x|D = 1] = \frac{e^{-\frac{y^2}{2\sigma^2}}}{\sum_{y'\in Z} e^{\frac{-y'^2}{2\sigma^2}}}$$

4 Sampling algorithms

In this section we will present the sampling algorithms used to sample from the discrete Gaussian as shown in the paper.

This first algorithm's point is to sample the wanted distribution by using less computational methods.

Algorithm 1 Sampling $Bernoulli(e^{-\gamma})$

Input : $\gamma \geq 0$
Output : One sample from $Bernoulli(e^{-\gamma})$

if $\gamma \in [\,0, 1]$ **then**
  $K \leftarrow 1$
  loop
    Sample $A \leftarrow Bernoulli(\gamma/K)$.
    if A $= 0$ **then** break the loop
    if A $= 1$ **then** set $K \leftarrow K + 1$ and continue the loop
  if K is odd **then return** 1
  if K is even **then return** 0
else
  for $k := 1$ **to** $\lfloor\gamma\rfloor$ **do**
    Sample $B \leftarrow Bernoulli(e^{-1})$
    if $B = 0$ **then** break the loop and **return** 0
    $C \leftarrow Bernoulli(e^{(\lfloor\gamma\rfloor-\gamma))})$
  return C

This second algorithm's role is to sample the Discrete Laplace distribution with precision and efficiency. And the third and last algorithm is the sampling from the Discrete Gaussian distribution that requires $O(1)$ operations on average.

Algorithm 2 Sampling for a Discrete Laplace

Input : $s, t \in Z, s, t \geq 1$.
Output : One sample from $Lap_Z(t/s))$
loop
Sample $U \in \{0, 1, \ldots, t-1\} uniformly at random.$
Sample $D \leftarrow Bernoulli(e^{-U/t})$.
if $D = 0$ **then** reject and restart.
Initialize $V \leftarrow 0$
loopSample $A \leftarrow Bernoulli(e^{-1})$.
if $A = 0$ **then** break the loop.
if $A = 1$ **then** set $V \leftarrow V + 1$ and continue.
Set $X \leftarrow U + tv$.
$Y \leftarrow \lfloor X/s \rfloor$
Sample $B \leftarrow Bernoulli(1/2)$.
if $B = 0$ and $Y = 0$ **then** reject and restart.
return $Z \leftarrow (1 - 2B)Y$.

Algorithm 3 Sampling for a Discrete Gaussian

Input : $\sigma^2 > 0$.
Output : One sample from $\mathcal{N}_Z(0, \sigma^2)$.
Set $t \leftarrow \lfloor \sigma \rfloor + 1$
loop
Sample $Y \leftarrow Lap_Z(t)$.
Sample $C \leftarrow Bernoulli(e^{\frac{-(|Y| - \sigma^2/t)}{2\sigma^2}})$.
if $C = 0$ **then** reject and restart.
if $C = 1$ **then return** Y.

Here we provide the python implementations of the above algorithms taken from the author's publically avaible code repository. To implement these algorithms, we first suppose we have two sampling algorithm for the Bernoulli and Uniform distributions.
The first algorithm is coded in one function.

```
# The following code is the python version of the Sampling Algorith of a
   Discrete Gaussian
import numpy as np
import math
from fractions import Fraction
import sample_bernoulli, sample_uniform
# We suppose that we have 2 sample algorithms sample_bernoulli and
   sample_uniform that are classicaly defined
# Alg 1
def sample_bernoulli_exp(gamma):
    gamma_frac = Fraction(gamma)
    if gamma <= 1 and gamma >= 0:
        k = 1
        a = sample_bernoulli(gamma_frac/k)
        while a != 1:
            k += 1
            a = sample_bernoulli(gamma_frac/k)
            if a == 0:
                break
        if k % 2 == 0:
            return 1
        elif k % 2 == 1:
            return 0
    elif gamma > 1:
        gamma_counting = Fraction(gamma)
        for k in range(math.floor(gamma)):
            b = sample_bernoulli_exp(Fraction(1))
            gamma_counting = gamma_counting - 1
            if b == 0:
                return 0
        c = sample_bernoulli_exp(gamma_counting)
```

The second algorithm needs the prior definition of the sample of a Geometric with exponential inside.

```
# Alg 2
def sample_geometric_exp(x):
    t = x.denominator
    u = sample_uniform(t)
    b = sample_bernoulli_exp(Fraction(u,t))
```

```
    while b !=1:
        u = sample_uniform(t)
        b = sample_bernoulli_exp(Fraction(u,t))
    v = 0
    a = Fraction(1,1)
    while sample_bernoulli_exp(a) != 1:
        v += 1
    value = v*t+u
    return value//x.numerator
def sample_discrete_laplace(lambd):
    lambda_frac = Fraction(lambd)
    while b !=1 or y != 0:
        b = sample_bernoulli(Fraction(1,2))
        y = sample_geometric_exp(1/lambda_frac)
    return y*(1 - 2*b)
```

The third and final algorithm needs the simple $floorsqrt()$ function that algorithmically calculates the floor of the square root.

```
# Alg 3
def floorsqrt(x):
    a = 0
    b = 1
    while b * b <= x:
        b = 2 * b
    while a + 1 < b:
        # this is a binary search
        c = (a + b)//2
        if c * c <= x:
            a = c
        else:
            b = c
    return a
def sample_discrete_gauss(sigma2):
    sigma2_frac = Fraction(sigma2)
    t = floorsqrt(sigma2)+1
    sample_y = sample_discrete_laplace(t)
    sample_c = ((abs(sample_y) - sigma2_frac/t)**2)/(2*sigma2_frac)
    while sample_bernoulli_exp(sample_c) != 1:
        sample_y = sample_discrete_laplace(t)
        sample_c = ((abs(sample_y) - sigma2_frac/t)**2)/(2*sigma2_frac)
    return sample_y
```

5 Comparing the discrete gaussian and the discrete Laplace's performance

The paper compares the properties of the discrete gaussian to it's more commonly used alternative: the discrete Laplace. Which is defined by:
Let $t > 0$. *The discrete Laplace distribution with scale parameter t is denoted* $Lap_Z(t)$.
It is a probability distribution supported on the integers and defined by :

$$\forall x \in Z, \underset{X \leftarrow Lap_Z(t)}{\mathbf{P}}[X = x] = \frac{e^{1/t} - 1}{e^{1/t} + 1}$$

The properties of the discrete Laplace are as follows:

1. Differential privacy:
   Let $q : \mathcal{X}^n \to Z$ *satisfy* $|q(x) - q(x')| \leq \Delta$ *for all* $x, x' \in \mathcal{X}^n$ *differing on a single entry. Define a randomized algorithm* $M : \mathcal{X}^n \to Z$ *by* $M(x) = q(x) + Y$ *where* $Y \leftarrow Lap_Z(\Delta/\epsilon)$. *Then M satisfies*$(\epsilon, 0)$*-differential privacy*

2. Utility: Qualitatively, in terms of utility, the discrete Laplace has sub-exponential tails.

3. Sampling: The sampling algorithm for this distribution is shown in the previous part of this report.

The authors explain how a quantitative comparison of these two distributions in the privacy context could be misleading. But the discrete gaussian still offers a huge advantage over the Laplace as they explain how in privacy contexts, it's often the composition of queries that is an object of interest and not one query by itself. And as we've explained in the previous section: this composition of queries is better expressed in the case of the discrete gaussian distribution.

Bibliography

[1] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. *The Discrete Gaussian for Differential Privacy*. 2020. DOI: 10.48550/ARXIV.2004.00010. URL: https://arxiv.org/abs/2004.00010.

[2] Ilya Mironov. "On significance of the least significant bits for differential privacy". In: *CCS '12: Proceedings of the 2012 ACM conference on Computer and communications security* (2012), p. 650.