Differential Privacy Synethic Data Generation using WGAN

Team: UCLANESL
Mani Srivastava, Moustafa Alzantot*

May 1st, 2019

1 Introduction

In this document, we provide a description and proof of privacy of the undergoing submission by team UCLANESLin the NIST: Differential Privacy Synthetic Data Challenge (Match 3) [1]. The goal of the challenge competition is to produce Differentially private synthetic datasets that preserve the utility of the original datasets on different tasks like using them to train clustering and classification models. The generation of the synthetic dataset should satisfy differential privacy guarantees so that it does not leak entries from the original dataset. Such a method for utility preserving synthetic dataset generation with differential privacy guarantees would be of an extreme value to provide researchers with datasets to work with when the original datasets are sensitive to the privacy of the individuals who have contributed to it or when regulations prohibit the release of these data such medical records datasets.

Our team submission combines together two state-of-art methods for synthetic data generation and accounting of differential privacy loss. Namely, the WGAN[4] method for training GAN (generative adversarial networks) models and the moments accountant [2] approach for calculating the bound of the privacy loss.

Our approach to solving the competition challenge is the following: We use the provided real dataset to train a generative model (GAN) using the improved Wasserstein GAN training algorithm. After training, the generator model of our GAN can be used to produce synthetic dataset by feeding noise samples from a prior distribution (e.g. normal distribution) into it. Compared to the original GAN formulation, the WGAN produces higher quality results and it is much more stable to train. In order to satisfy the differential privacy constraints, the training of our WGAN models is made private by sanitizing the gradient before applying gradient descent steps. Sanitization of gradient occurs in two

*Corresponding Author: Moustafa Alzantot (malzantot@ucla.edu), TopCoder handle: m_alzantot

steps: first, the l_2 norm of the computed gradient is clipped to ensure that the gradient function has bounded l_2 sensitivity. Second, we apply the Gaussian mechanism by adding Gaussian noise to the sum of clipped gradient. During the model training, we track the total privacy loss by using the moments accountant method [2].

The rest of this document is organized as follows: Section 2 provides a brief background on both Wasserstein GANs and differential privacy. Section 3 describes our differential private training algorithm of WGAN model. Section 4 proves that our training algorithm satisfies the differential privacy requirements.

2 Background

2.1 Generative Adversarial Networks

Deep learning generative models Deep learning models using neural networks have achieved huge success in many tasks (e.g. image recognition, object detection, speech recognition, etc.). Deep learning models can be categorized as either discriminative models or generative models. Discriminative models attempt to learn the mapping between inputs and the target variables. Formally, they learn to compute $P(y|x;\theta)$. Where x is the model input, y is the output and θ denotes the model parameters. On the other hand, generative models attempt to learn the joint distribution between the input and output variables i.e. $P(x, y; \theta)$.

The first wave of deep learning advancements in the past decade has been focused on discriminative models (e.g. image recognition and speech classification) because they are easier to train and achieve good results on. While training generative models was still considered relatively harder to make progress on. Especially when the input data dimensions are large then training a generative model using maximum likelihood needs huge amounts of training data (due to the curse of dimensionality) and involves approximations of many intractable probabilistic computations that lead to poor quality results.

Generative Adversarial Networks A breakthrough happened in 2014 when researchers introduced a new approach known as *Generative Adversarial networks* (GANs) [10, 9]. The goal of the training is to learn a probability distribution $p_g(\mathbf{x};\theta_G)$ that matches the distribution of input training data $p_{data}(\mathbf{x})$. While GANs do not provide an explicit representation of the learned probability distribution, we can use a GAN to generate samples from the distribution it has learned. This is useful in the case when the goal is to generate synthetic dataset which makes them a suitable technique for the purpose of our competition.

The basic idea of adversarial networks is to maintain two neural network models D, and G that compete against each other. The $D(\mathbf{x};\theta_D)$ model is termed the *discriminator* and accepts an input that is either coming from the real data distribution $\mathbf{x} \sim p_{data}(\mathbf{x})$ or from the outputs of the generator model $\mathbf{x} \sim p_g(\mathbf{x};\theta_G)$. The objective of model D training is to increase its accuracy in

making a difference between inputs that are coming from the real data distribution and those that are coming from the generator model outputs. In original GAN[10] formulation the output of D model is a probability value in the $[0, 1]$ range.

The generator model $G(\mathbf{z}; \theta_G)$ takes an input noise vector sampled from a given prior distribution (e.g. normal or uniform) to produce fake samples that mimic the samples in the real training dataset. The objective of the model G training is to confuse the model D and limits its ability to make distinction between real examples and generator outputs.

Formally, G and D play the following min-max game with value function $V(D, G)$

$$\min_G \max_D \; V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[log\, D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z})))]$$

When both G and D are differentiable functions (e.g. neural networks), the two models are training algorithm alternates between updating D, and G and the parameters of each model are updated using gradient descent and backpropagation.

It has been empirically found [9] that it is better to train model G to maximize $\mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[logD(G(\mathbf{z}))]$ instead of minimizing $\mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z})))]$ in order to provider stronger gradient updates to improve the θ_G parameters in the early stages of training. This technique is known as the log D trick. One important aspect of the GAN training approach: only the model D is trained directly using examples from the real dataset, while G model is trained through the feedback it receives from the D model outptus on the samples it generates.

Wasserstein GAN Although GANs constitute a significant improvement than previous methods for training generative models, researchers faced a major challenge in scaling up GANs and stabilizing their training on big and high dimensional datasets. GANs suffered from instability in training that prevented the competing models D, and G from reaching the Nash equilibrium of the min-max game so that both model the quality of D and G oscillates.It has also been found that when G model produces samples that are far from the true data samples, the discriminator saturates and results in vanishing gradient that limits the chances of G model to improve. This problem has motivated researchers to develop many alternatives and tricks to make GAN training more stable and improve the equality of results[16, 15, 3, 17, 14].

A notable fix to GAN adversarial training stability issues and improve the results quality is the Wasserstein GAN [4]. In the original GAN formulation, the generator attempts to minimize the Jenson Shannon Divergence (JSD) between p_g and p_d. Rather than minimizing the JSD, Wasserstein GAN changes the GAN training loss so that it minimizes an efficient approximation of the Earth Movers (EM) distance between the the two distributions $p_{data}(\mathbf{x})$, and $p_g(\mathbf{x}; \theta_G)$.

The Earth-Mover distance of Waserstein-1 distance is defined as:

$$W(P_{data}(\mathbf{x}), P_g(\mathbf{x};\theta)) = \inf_{\gamma \in \Pi(P_{data}(\mathbf{x}), P_g(\mathbf{x};\theta))} \mathbb{E}_{(x,y)\sim\gamma}\left[\, \|x-y\| \,\right], \qquad (1)$$

where $\Pi(P_{data}(\mathbf{x}), P_g(\mathbf{x};\theta))$ is the set of all joint distributions $\gamma(x,y)$ whose marginals are respectively $P_{data}(\mathbf{x})$ and $P_g(\mathbf{x};\theta)$. Intuitively, the EM distance is the "cost" of the optimal transport plan between the two distributions where $\gamma(x,y)$ represents how much "mass" must be transported from x to y in order to transform the distributions $P_{data}(\mathbf{x})$ into the distribution $P_g(\mathbf{x};\theta)$.

EM distance has a number of appealing properties than the JSD that leads to an improved and more stable training of the WGAN models. The D model in W-GANs is termed as *critic* instead of *discriminator*, because its output value is no longer required to represent a probability value and therefore no longer constrained to be within the $[0, 1,]$ range. Unlike the JSD distance, EM distance does not saturate when the D model produces good results and that allows WGAN model to train D model till optimality and after that, it can be used to provide useful gradient to train the generator model. Therefore, it is no longer needed to put a lot of effort maintaining a balance between D and G while training which was a reason for a lot of troubles in the original GAN training. Instead, in $WGAN$ the better the critic D, the better gradients it can provide as feedback to train the generator G.

To avoid intractable infimum operation in EM distance definition, WGAN[4] employs (the Kantrovitch-Rubinstein duality [18]) to compute it with following formula instead:

$$W(P_{data}(\mathbf{x}), P_g(\mathbf{x};\theta)) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim P_{data}(\mathbf{x})}[f(x)] - \mathbb{E}_{x\sim P_g(\mathbf{x};\theta)}[f(x)] \qquad (2)$$

Therefore, if we have a family of functions $w \in \mathcal{W}$ that are all K-Lipschitz for some K, the following equation becomes equal to Wasserstein distance multiplied by a multiplicative factor K.

$$\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim P_{data}(\mathbf{x})}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(G(z;\theta))]$$

This equation can be approximated when the neural network parameters w are lying in a compact space. In order to ensure that parameters lie in a compact space, WGAN[4] enforces this by clamping the model weights to a fixed box (say $\mathcal{W} = [-0.01, 0.01]^l$) after each gradient update.

The min-max game between the two networks becomes :

$$\min_{\theta} \max_{w\in\mathcal{W}} \mathbb{E}_{x\sim P_{data}(\mathbf{x})}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(G(z;\theta))]$$

The WGAN training algorithm is provided in 1.

In Algorithm 1, f_w is the critic model function with parameters w, and G is the generator function with parameters θ. The critic is trained multiple for multiple steps (n_{critic}) before training the generator in order to reach its optimality and improve the gradients it can provide to improve the generator.

Algorithm 1 Algorithm for training WGAN

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

1: **while** θ has not converged **do**
2: **for** $t = 0, ..., n_{\text{critic}}$ **do**
3: Sample $\{x^{(i)}\}_{i=1}^{m} \sim P_{data}(\mathbf{x})$ a batch from the real data.
4: Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
5: $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(G(z^{(i)};\theta))\right]$
6: $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
7: $w \leftarrow \text{clip}(w, -c, c)$
8: **end for**
9: Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
10: $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(G(z^{(i)};\theta))$
11: $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: **end while**

The weight clipping of critic parameters w in line 7, ensure that the weights parameters lie in a compact space and the $RMSProb$[11] is an extension to gradient descent that adaptively adjusts the learning rate for each dimension in the weight vector.

2.2 Differential Privacy

Differential privacy is a privacy model that provides privacy guarantees for algorithms on aggregate databases. Intuitively, it provides a promise made by the data holder to the data subject: "you will not be affected by allowing your data to be used in the released results of statistical queries no matter what other auxiliary information the attacker may have".

Differential Privacy : An randomized algorithm $\mathcal{M} : \mathcal{D}^n \to \mathcal{R}$ is (ϵ, δ)-differentially private if for all neighbouring databases d, d' such that $||d - d'|| \leq 1$:

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

for all $S \in \mathcal{R}$. Typically δ is less than the inverse of any polynomial in the size of the database. If $\delta = 0$, then $\mathcal{M}$ is ϵ-differentially private. The smaller the value of ϵ, the stricter privacy guarantee is offered by $\mathcal{M}$.

Post-Processing property [7] if $\mathcal{M} : \mathcal{D}^n \to \mathcal{R}$ is randomized algorithm taht is an ϵ-differentially private mechanism by using:-differentially private. Let $f : \mathcal{R} \to \mathcal{R}'$ be an arbitrary data-independent randomized algorithm. Then $f \circ \mathcal{M} : \mathcal{D}^n \to \mathcal{R}'$ is an (ϵ, δ)-differentially private.

Laplace Mechanism [7] Any given function $f : \mathcal{D}^n \rightarrow \mathcal{R}$ can be approximated via an ϵ-differentially private mechanism by using:

$$\mathcal{M}(x) = f(x) + Lap(\frac{\Delta_f}{\epsilon})$$

where $Lap(\sigma)$ is a value sampled from Laplace distribution with zero mean and σ scale, and Δ_f is the sensitivity of f defined as

$$\Delta_f = \max_{\substack{x,y \in \mathcal{D}^n \\ ||x-y||=1}} ||f(x) - f(y)||_1$$

Gaussian Mechanism[7] Another alternative to adding Laplacian noise is to add Guassian noise. In this case rather than scaling noise to l_1 sensitivity Δ_f, we instead scale the noise to the l_2 sensitivity $\Delta_2(f)$. Therefore, any given function $f : \mathcal{D}^n \rightarrow \mathcal{R}$ can be approximated via an ϵ-differentially private mechanism by using:

$$\mathcal{M}(x) = f(x) + \mathcal{N}(0, \Delta_2^2(f) \cdot \sigma^2)$$

where $\Delta_2(f)$ is l_2 sensitivity of function f

$$\Delta_2(f) = \max_{\substack{x,y \in \mathcal{D}^n \\ ||x-y||=1}} ||f(x) - f(y)||_2$$

A single application of Guassian mechanism to a function f of l_2 sensitivity $\Delta_2(f)$ satisfies (ϵ, δ) differential privacy if $\sigma \geq \frac{\sqrt{2 \log \frac{1.25}{\delta}}}{\epsilon}$

Sequential Composition and Privacy Accountant [13] : The sequential composition property of differential privacy states that for any sequence of computations that each of them is differentially private in isolation will also provide differential privacy. If each of $\mathcal{M}_i$ is ϵ_i differentially-private then the sequence of $\mathcal{M}_i$ provides $\sum_i \epsilon_i$-differential privacy. [13] proposes the use of **privacy accountant** to maintain track of the total privacy loss in complex algorithms that consist of individual differentially-private computation steps.

Strong Composition Theorem: The lemma 2.3[5, 8] can be used to compute the total privacy loss of T steps of adaptive composition. When $\epsilon, \delta' \geq 0$, The class of ϵ-differentially private algorithms satisfies (ϵ', δ')-differential privacy under T-fold adaptive composition for $\epsilon' = \sqrt{2 * T \ \ln(\frac{1}{\delta'})}\epsilon + T\epsilon(e^\epsilon - 1)$

Moments Accountant Method: Like privacy accountant method, the moments accountant also keeps track of the privacy loss for sequence of computations. However, the moments accountant uses the tail bound of log moments of privacy loss random variable to obtain a more strict bound for privacy loss.

For any two neighbouring data-sets $d, d' \in \mathcal{D}^n$, a mechanism $\mathcal{M}$, auxiliary input **aux**, and an outcome $o \in \mathcal{R}$, the privacy loss at o is defined as

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

where the **aux** input for the k^{th} mechanism $\mathcal{M}_{\|}$ includes the output of all previous mechanisms. Therefore, allowing for composition of mechanisms.

For any given mechanism, the λ^{th} moment $\alpha_{\mathcal{M}}(\lambda; \mathbf{aux}, d, d')$ is the log of the moment generating function evaluated at value λ

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

If we let $\alpha_{\mathcal{M}}(\lambda)$ to be the maximum taken over all possible **aux** and all neigbhouring databases d, d'

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

Then $\alpha_{\mathcal{M}}$ has two important properties:

- **Composability**: suppose a mechanism $\mathcal{M}$ consists of sequence of adaptive mechanisms $\mathcal{M}_1, ...\mathcal{M}_k$ where $M_i : \Pi_{j=1}^{i-1} \mathcal{R}_j \times \mathcal{D} \rightarrow \mathcal{R}_i$, then for any λ
$$\alpha_M(\lambda) \leq \sum_{i=1}^{k} \alpha_{\mathcal{M}i}(\lambda)$$
- **Tail bound**: For any $\epsilon > 0$, the mechanism $\mathcal{M}$ is (ϵ, δ)-differentially private for
$$\delta = \min_{\lambda} \exp(\alpha_{\mathcal{M}}(\lambda) - \lambda\epsilon)$$

The proof for the two properties is provided in [2].

As a result, we can compute the overall bound $\alpha_{\mathcal{M}}$ by summing the bounds of each step $\alpha_{\mathcal{M}_i}$. Then, we can use the tail bound property to convert the moments bound to (ϵ, δ)-differential privacy guarantee.

3 Our approach

Algorithm 2 describes the procedure for training our differentially-private WGAN.

Compared to the original WGAN training algorithm given in 1, we add the following steps:

Compute Per-Example Gradient: Rather than computing the mini batch gradient value, we need to compute the gradient of loss function with respect to the *critic* parameters per each example $x^{(j)}$ in the training data $g_w(x^{(j)})$. $x^{(j)}$

Gradient Norm Clipping: Since choosing the amount of noise we add while applying the Gaussian mechanism depends on the l_2 sensitivity of the

Algorithm 2 Algorithm for training (ϵ, δ)-differentially private WGAN

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration. Noise scale σ, group size L, Gradient Norm bound c_p

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

for $t \in [T]$ **do**
  for $i = 0, ..., n_{\text{critic}}$ **do**
    Pick a random sample $L_{t,i} = \{\mathrm{x}^{(j)}\}_{j=1}^{L} \sim P_{data}(\mathbf{x})$ from the real data.
    Sample $\{z^{(j)}\}_{j=1}^{L} \sim p(z)$ a batch of prior samples.
    Compute the per-example gradient
    $g_w(x^{(j)}) = \nabla_w f_w(x^{(j)})$ for $\mathrm{x}^{(j)} \in L_{t,i}$
    $g_w(z^{(j)}) = \nabla_w f_w(G(z^{(j)}; \theta))$ for $j \in [L]$.
    Clip gradients
    $\bar{g}_w(x^{(j)}) = g_w(x^{(j)}) / max(1, \frac{||g_w(x^{(j)})||_2}{c_p})$ for $x^{(j)} \in L_{t,i}$
    $\bar{g}_w(z^{(j)}) = g_w(z^{(j)}) / max(1, \frac{||g_w(z^{(j)})||_2}{c_p})$ for $j \in [L]$
    Add Noise
    $\tilde{g}_w = \frac{1}{L}\left(\sum_{j=1}^{L} \bar{g}_w(x^{(j)}) + \mathcal{N}(0, \sigma^2 c_p^2 \mathbf{I})\right) - \frac{1}{L}\sum_{j=1}^{L} \bar{g}_w(z^{(j)})$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, \tilde{g}_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(j)}\}_{j=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{j=1}^{m} f_w(G(z^{(j)}; \theta))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end for

input function. We clip the l_2 norm of each gradient so that the maximum gradient norm becomes c_p.

Adding Noise: We apply Gaussian mechanism to santize the gradients of individual examples by adding noise $\mathcal{N}(0, \sigma^2 c_p^2 \mathbf{I})$ to the sum of clipped per-example gradients. Therefore, the aggregated gradient for each step becomes differential private. In order to achieve (ϵ, δ) differential privacy for each step, it suffice to chose σ to be $\geq \frac{\sqrt{2 \log \frac{1.25}{\delta}}}{\epsilon}$[7].

Using Moments Account: We use the moments accountant to track the privacy loss of the whole training algorithm by through composition of bounds of the log of moments of privacy loss per each step.

4 Proof of Privacy

Our proof of privacy for algorithm 2 follows from the following:

Differential Privacy Proof for Critic:

To ensure that *critic* is differentially-private, we changed the WGAN training algorithm 1 into algorithm 2 by adding the following steps:

- We apply gradient clipping to bound the l_2 sensitivity of the gradient computed for each example.
- We then add noise to the sum of the clipped gradients using the Gaussian mechanism. In above step, we bound the per-sample gradient to have a maximum l_2 norm equal to c_p. This ensures that the sum of clipped gradients will also be l_2 bounded for any two neighbouring data-sets (using Cauchy-Schwartz inequality) with the norm bound equal to c_p (assuming bounded differential privacy).
- We use the moments accountant method to select the proper scale of Gaussian noise that we add to the gradients.

By bounding the l_2 sensitivity of the gradient updates and applying Gaussian mechanism, each update step becomes (ϵ, δ)-differentially private when the noise scale σ is at least $\frac{\sqrt{2 \log \frac{1.25}{\delta}}}{\epsilon}$[7] with respect to the training lot. Since the lot itself is a random sample of the training database, the privacy amplification theorem [12, 6] states that each step will be $(O(q\epsilon), q\delta)$-differentially private with respect to the training database where sampling ratio $q = \frac{L}{N}$.

Using the moments accountant[2] we can compute the total privacy loss of the training algorithm to show that our final model is $(O(q\,\epsilon\,\sqrt{T\,n_{critic}}), \delta)$-differentially private. **The proof of this bound can be found in [2]**.

When (ϵ, δ)-privacy budget is very small. We prefer to use a privacy accountant based on the strong composition theorem [5] rather than the moments accountant [2]. Even though the bounds of strong-composition theorem accumulate faster, it allows us to run training for few steps before running out of privacy budget. **The proof of correctness the privacy-bounds computed by strong composition theorem can be found in [5] (Lemma 2.3)**.

Differential Privacy Proof for Generator:
In WGAN training, only the *critic* parameters w are updated directly using the real dataset entries. While the generator parameter θ updates are updated as a function of the critic output values. Therefore, if the critic itself is differentially private, the generator will also be differentially private **(Using the post-processing invariance property of differential privacy[7])**.

5 Tracking privacy Loss

In our implementation, we use the privacy accountant method of [2], to keep track of the accumulated privacy loss spending after each training step that updates the *critic* weight values. If the accumulated privacy loss spent exceeds the maximum allowed limits of (ϵ, δ), then we abort the training early and don't perform any further updates to the *generator* model.

We employ two state-of-art methods to implement a privacy accountant that tracks the total privacy budget spent after each step of training our models.

- **moments accountant** [2] which tracks privacy loss using the tail bound of the log of moments of privacy loss random variable.
- **amortized accountant** which uses privacy amplification theorem [6, 12] and strong composition theorem[5, 8] to provide an amortized bound of privacy loss.

Both methods are known to provide the state-of-art tight bounds of privacy loss tracking. Although the moments accountant [2] provides a tighter bound. Although the growth of privacy loss calculated with moments accountant is much less than the growth of privacy loss calculated by strong-composition theorem, We have found that when ϵ_{total} is very small, the bounds computed by strong composition theorem are actually less than the bounds of moments accountant. Therefore, our calculations of privacy loss uses the **amortized accountant** when privacy loss $\epsilon \leq 0.7$, otherwise we use the **moments accountant**.

6 Running our solution

Our implementation uses the `TensorFlow` open-source deep learning framework which is more efficiently run using GPU enabled machines using. It can be also run in a container using `NvidiaDocker`. The runtime is signficantlly improved when running on a GPU- powered machine. It has been tested to work successfully on a machine with Titan X GPU.

Steps to install `NvidiaDocker` can be found in `https://github.com/NVIDIA/nvidia-docker`

After unziping the given zip file. First, build the docker image.

```
docker build -t uclanesl_wgan .
```

then, run the image while giving the appropriate argument values.

then, run the image while giving the appropriate argument values.

```
docker run --runtime=nvidia -v $(pwd):/x uclanesl_wgan /
    x/data.csv /x/output.csv /x/data-specs.json  epsilon
    [delta]
```

where

data.csv: input dataset file.

output.csv: output file.

data-specs.json: metadata file.

epsilon ϵ privacy budget.

delta: **(optional)** value of δ parameter. If it is not given, then δ is chosen to be $\frac{1}{n^2}$ where n is the size of private dataset.

7 Data pre-processing and post-processing

7.1 Pre-processing

In order to prepare the input data for our model training, we apply the following transformations for the input data columns. Each column is considered as one the three following types:

- **Categorical**: Categorical columns are pre-processed by applying a one-hot encoding to their values. The transformation happens through two steps:

  1. Assume the column has N distinct values, then we map the values for the columns to values in the range $\{0, 1, ..., N-1\}$. This process is needed to prepare the values for the next step of one-hot-encoding. The number of distinct values N for each columns is identified using either of the following ways:
     - From the **data-specs.json** metadata file. We use the *maxval* property to identify the maximum value of the column then the number of distinct values is equal to $maxval + 1$.
     - From the codebook **codebook.cbk** file that was included in the competitor pack with description of the possible set of values for each column.
     - For the **MIGSEA5** and **OCC** columns, due to the high sparsity of the column values we identify the set of distinct values from the tables available on IPUMS website:
       `https://usa.ipums.org/usa/volii/seacodes.shtml`
       `https://usa.ipums.org/usa/volii/occ1940.shtml`
     - For the set of geographic state-dependent columns (**SEA, METAREA, METAREAD, COUNTY, CITY**), we collect the number of distinct values from the input data.
  2. Apply one hot encoding to the column value.

- **numeric (*integer*)**: numeric columns are used as it is without any transformations.

- **numeric or not-available**: for columns which have 'not available' (i.e. 99999) codes we transform each column into two columns which are : a boolean column which denotes which the column value is available or not. A numeric column which is equal to the input column value if it is available (i.e. not equal to the 9999 code) or a zero otherwise.

7.2 Post-processing

After sampling synthetic data from the generator model, we apply post-processing step to format the data in the same format as input data. The post-processing

step does not require information from the input data therefore it satisfies the **post-processing in-variance property of differential privacy**.

Post-processing takes place by either or the following ways depending on the column type:

- **Categorical**: For categorical columns, we treat the generator output as a probability distribution over the set of distinct column values. The post processing happens in two steps

  1. We randomly sample a value according the softmax distribution produced by the generator. This produces an integer value in the [0, N-1] range where N is the number of distinct values for the data column.
  2. We map the [0, N-1] values to the set of distinct values for the data column in the inverse direction of mapping that we did during the pre-processing. *mapping between [0, N-1] range and the set of distinct values was computed in pre-processing step.*

- **numeric (*integer*)**: numeric columns are used as it is without any transformations. We clip the column value to the [0, *maxval*] range where *maxval* is identified from the metadata **data-specs.json** file.

- **numeric or not-available**: for these columns, the generator model produces a boolean value specifying whether the column has a valid value or *not-available*. If that value is true, then the post-proceed value is *maxval*. The second output is a numeric value. The numeric value is clipped to [0, *maxval*] range and if the boolean value

8 Using public data

Our solution does not use any public datasets to training the generator. Once the generator model is trained from input data using differentially-private training algorithm. The generator model is capable of producing new synthetic data.

- We assume that the metadata of each column (i.e. data provided in **data-specs.json**) such the maximum value of each column are to be considered public information.

- Also we also consider as public information, the **codebook.cbk** provided in the competitor pack and we use it to identify the set of possible values for categorical columns while doing data pre-processing.

- For the **MIGSEA5** and **OCC** columns, we identify the set of possible values (but not the column values distribution) from IPUMS website : `https://usa.ipums.org/usa/volii/seacodes.shtml` and `https://usa.ipums.org/usa/volii/occ1940.shtml`.

- Since the final testing dataset will be for different states other than Colorado which was used during the provisional testing phase, As posted in the competition forum (`https://apps.topcoder.com/forums/?module=Thread&threadID=935216`) for the geographic state-dependent columns (**SEA**, **METAREA**, **METAREAD**, **COUNTY**, **CITY**), we collect the set of distinct values for each of these columns from the input data, but we don't collect any information about the distribution of values.

9 Conclusion

In this document, we have provided a proof for the differential privacy of the UCLANESL team submission to the NIST Synthetic data challenge (**Match 3**).

References

[1] Differential privacy synthetic data challenge. `https://challenge.gov/a/buzz/nist-pscr/differential-privacy-synthetic-data-challenge`.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[3] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[5] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Differentially private empirical risk minimization: Efficient algorithms and tight error bounds. *arXiv preprint arXiv:1405.7085*, 2014.

[6] Amos Beimel, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. In *Theory of Cryptography Conference*, pages 437–454. Springer, 2010.

[7] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[8] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE, 2010.

[9] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, page 14, 2012.

[12] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[13] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.

[14] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[17] Akash Srivastava, Lazar Valkoz, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3308–3318, 2017.

[18] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.