Differentially Private Normalizing Flows for Synthetic Tabular Data Generation

Jaewoo Lee[1], Minjung Kim[2], Yonghyun Jeong[2], Youngmin Ro[2]

[1]University of Georgia
[2]Samsung SDS
jwlee@cs.uga.edu, {mj100.kim, yhyun.jeong, youngmin.ro}@samsung.com

Abstract

Normalizing flows have shown to be a promising approach to deep generative modeling due to their ability to exactly evaluate density — other alternatives either implicitly model the density or use approximate surrogate density. In this work, we present a differentially private normalizing flow model for heterogeneous tabular data. Normalizing flows are in general not amenable to differentially private training because they require complex neural networks with larger depth (compared to other generative models) and use specialized architectures for which per-example gradient computation is difficult (or unknown). To reduce the parameter complexity, the proposed model introduces a conditional spline flow which simulates transformations at different stages depending on additional input and is shared among sub-flows. For privacy, we introduce two fine-grained gradient clipping strategies that provide a better signal-to-noise ratio and derive fast gradient clipping methods for layers with custom parameterization. Our empirical evaluations show that the proposed model preserves statistical properties of original dataset better than other baselines.

Introduction

Differentially private generative modeling of data has received much attention due to the ability to generate samples from the learned distributions while protecting privacy. However, majority of existing approaches focused on two popular deep generative models: generative adversarial networks (GANs) and variational autoencoders (VAEs), and other types of generative models remain unexplored. In this paper, we develop a differentially private normalizing flow model, called DP-HFlow, to synthesize realistic tabular data.

Normalizing flows (NFs) (Tabak and Turner 2013; Rezende and Mohamed 2015) allow exact evaluation of likelihood via change of variables and hence can be directly trained with maximum likelihood estimation (MLE). Despite these advantages, modeling tabular data using a flow-based model under differential privacy poses several challenges. First, many real-world tabular datasets contain both continuous and discrete variables. Unfortunately, NFs are not immediately applicable to datasets with mixed types. When training a continuous density model, such as normalizing flows, on discrete data using MLE, the model may end up learning a degenerate distribution in which arbitrarily high likelihoods are assigned to few particular values (Uria, Murray, and Larochelle 2013). To avoid this, DP-HFlow first converts discrete values into continuous ones using variational dequantization (Ho et al. 2019) and applies the same density estimator to both continuous and discrete variables to capture the interactions between them.

Second, compared to other generative models, NF models have a higher parameter complexity. To improve the expressiveness, flow-based models are normally constructed by composing multiple sub-flows. In practice, this is implemented by repeating the same blocks of transformations. For example, GLOW (Kingma and Dhariwal 2018) model contains 32 blocks of actnorm, 1×1 convolution, and affine coupling transformations. This increased complexity poses a significant challenge to the differentially private training of NF models. To reduce the parameter complexity, we propose a conditional spline flow whose transformation is conditionally defined by an additional input. The conditional spline flow is shared among multiple layers of sub-flows and hence helps mitigate the parameter complexity.

Third, the neural networks in DP-HFlow used to generate the parameters of each sub-flow are *heterogeneous* in terms of both architecture and parameter complexity. As a result, the scales of their gradients vary significantly not only among different networks but also among layers. We observed that a naïve application of gradient clipping technique (Abadi et al. 2016) results in a poor signal-to-noise ratio for some layers, making the training difficult. In DP-HFlow, we introduce two advanced strategies for fine-grained gradient clipping (namely, *per-unit clipping* and *stochastic sparsification*) and show that they can accelerate private training by improving the signal-to-noise ratio.

Lastly, the gradient clipping technique requires modifying *per-example* gradients. For easy Jacobian determinant computation, NF models use neural network layers with *custom* factorized parameters. While per-example gradients for such layers can be computed using auto-differentiation library with gradient accumulation trick, it forces the model to process one example at a time instead of as a batch, which unacceptably slows down the training of differentially private models (Bagdasaryan, Poursaeed, and Shmatikov 2019). To improve the scalability of private training, we analyze the back-propagation equations for these custom layers and derive efficient clipping methods using the fast gradient clipping framework due to (Lee and Kifer 2021).

This paper makes the following contributions: (i) to the best of our knowledge, DP-HFlow is the first differentially private normalizing flow model that can handle datasets with mixed types of attributes; (ii) we examine the feasibility of training parameter-heavy models (such as normalizing flows) under differential privacy, identify issues, and propose conditional spline flow and two fine-grained gradient clipping methods; (iii) we introduce fast per-example gradient clipping methods for linear transformations popularly used in NFs. (iv) our extensive empirical evaluations show that DP-HFlow outperforms existing private generative models in terms of capturing statistical properties of input datasets.

Related Work

DP-SGD Many existing approaches rely on the DP-SGD framework introduced in (Abadi et al. 2016) to train deep neural networks under differential privacy. DP-SGD treats weight vectors of all layers as a single vector in $\mathbb{R}^n$ (i.e., layer agnostic) and clips the per-example gradient w.r.t. the concatenated weight vector using a clipping threshold C. (McMahan et al. 2018) introduced an advanced clipping strategy, called per-layer clipping, in which the global threshold C is first split between layers. The gradient vectors for different layers are clipped separately using local threshold values. Our work extends the idea of per-layer clipping and introduces more fine-grained clipping strategies. We found that this fine-grained clipping is essential for achieving good performance and for training heterogeneous networks in normalizing flows. The work closest to ours is (Waites and Cummings 2021) in which a masked autoregressive flow (MAF) model is trained for simple density estimation task by directly applying the DP-SGD framework. However, their model was trained only using continuous variables and discrete variables were ignored.

Differentially Private GANs (Xie et al. 2018) proposed a differentially private Wasserstein GAN and showed that the sensitivity of gradient can be bounded by clipping weights instead of gradients. (Chen, Orekondy, and Fritz 2020) also introduced a private version of Wasserstein GAN, called GS-WGAN. Based on the observation that only the generator needs to be released for sample synthesis, they proposed to sanitizes the gradient passed from the discriminator to the generator. In contrast, PATE-GAN (Jordon, Yoon, and van der Schaar 2019) aims to train a differentially private (student) discriminator using the PATE framework (Papernot et al. 2017). Although GAN-based models can generate high quality samples, their training process is known to be notoriously unstable (Chu, Minami, and Fukumizu 2020), and the models are susceptible to mode collapse issue.

Differentially Private VAEs VAEs (Kingma and Welling 2014) are latent variable models in which the encoder network approximates the posterior distribution of latent variable and the decoder learns to reconstruct the data from posterior samples. Compared to GANs, the training of VAEs is stable but they are trained by maximizing a surrogate loss, called evidence lower bound (ELBO), rather than actual likelihood. (Chen et al. 2018) trained a differentially private VAE model using the DP-SGD framework. In (Tantipongpipat et al. 2020), a variant of VAE that incorporates a discriminator and a generator network was introduced.

Kernel-based methods (Balog, Tolstikhin, and Schölkopf 2018; Harder, Adamczewski, and Park 2021) use kernel functions to represent a probability distribution as a point in a reproducing kernel Hilbert space (RKHS). (Harder, Adamczewski, and Park 2021) proposed DP-MERF which exploits the fact that computing the maximum mean discrepancy (MMD) between true dataset and synthetic dataset only requires the kernel matrix computed from the true dataset. Thus, DP-MERF releases the perturbed kernel matrix under differential privacy. Given the noisy kernel matrix, a generator is trained to minimize the MMD between the true and synthetic datasets.

Preliminaries

Differential Privacy

Two datasets D_1 and D_2 are said to be *neighboring* if D_1 can be obtained by adding or removing one record from D_2, and we write $D_1{\sim}D_2$ to denote this relationship. Differential privacy hides the influence of one individual by constraining the outputs of an algorithm over two neighboring datasets to be probabilistically indistinguishable.

Definition 1 ((ϵ, δ)-DP (Dwork et al. 2006b,a))**.** *Let $\epsilon \geq 0$ and $\delta \geq 0$. A randomized mechanism (algorithm) $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy if for all $S \subseteq \mathrm{range}(\mathcal{M})$ and for all pairs of neighboring datasets $D_1{\sim}D_2$,*

$$\Pr[\mathcal{M}(D_1) \in S] \leq \exp(\epsilon) \Pr[\mathcal{M}(D_2) \in S] + \delta \,.$$

The probability is with respect to the randomness in $\mathcal{M}$.

When training a neural network under differential privacy, a privacy mechanism is applied to the set of gradients computed from a minibatch to ensure privacy of all parameter updates. To accurately track the total privacy loss incurred during the processing of all minibatches, we use Rényi Differential Privacy (RDP) (Mironov 2017). After training finishes, the RDP guarantee can be converted to that of (ϵ, δ)-DP using the conversion result in (Mironov 2017, Proposition 3). RDP relies on the concept of Rényi divergence:

Definition 2 (Rényi Divergence)**.** *Let P_1 and P_2 be probability distributions over a set Ω and let $\alpha \in (1, \infty)$. Rényi α-divergence $\mathfrak{D}_\alpha$ is defined as:*

$$\mathfrak{D}_\alpha(P_1 \parallel P_2) = \frac{1}{\alpha - 1} \log(\mathbb{E}_{x \sim P_2}\left[P_1(x)^\alpha P_2(x)^{-\alpha}\right]) \,.$$

Rényi differential privacy requires two parameters: a moment α and a parameter ϵ that bounds the moment.

Definition 3 ((α, ϵ)-RDP (Mironov 2017))**.** *Given a privacy parameter $\epsilon \geq 0$ and an $\alpha \in (1, \infty)$, a randomized mechanism $\mathcal{M}$ satisfies (α, ϵ)-Rényi differential privacy (RDP) if for all D_1 and D_2 that differ on the value of one record, $\mathfrak{D}_\alpha(\mathcal{M}(D_1) \parallel \mathcal{M}(D_2)) \leq \epsilon$.*

Given a deterministic function f and its sensitivity, we can achieve (α, ϵ)-RDP by applying the Gaussian mechanism to f. The mechanism adds Gaussian noise scaled according to the sensitivity of f.

Definition 4 (L_2 sensitivity)**.** *Let L_2 sensitivity of f, denoted by Δ_f is equal to $\sup_{D_1 \sim D_2} ||f(D_1) - f(D_2)||_2$, where the supremum is taken over all pairs of neighboring datasets.*

Lemma 1 (Gaussian Mechanism (Mironov 2017))**.** *Let f be a function. Let $\alpha > 1$ and $\epsilon > 0$. Let M be the mechanism that, on input D, returns $f(D) + N(0, \sigma^2 \mathbf{I})$, where $\sigma^2 = \frac{\alpha \Delta_f^2}{2\epsilon}$. Then M satisfies (α, ϵ)-RDP.*

Normalizing Flows

Let $\mathbf{z} \in \mathbb{R}^d$ be a noise vector drawn from a simple probability distribution (called a base distribution) with a tractable density function $\pi(\mathbf{z})$ such as standard Gaussian $\mathcal{N}(0, \mathbf{I}_d)$, where $\mathbf{I}_d$ denotes $d \times d$ identity matrix. A normalizing flow is an invertible, differentiable transformation $\mathbf{g}_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$, parameterized by $\boldsymbol{\theta}$, that models the mapping between noise $\mathbf{z}$ and data $\mathbf{x}$, i.e., $\mathbf{z} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x})$ (or equivalently $\mathbf{x} = \mathbf{g}^{-1}(\mathbf{z})$). Using the change of variables formula, the probability density function of $\mathbf{x}$ can be expressed by

$$p(\mathbf{x}; \boldsymbol{\theta}) = \pi(\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x})) \left|\det J_{\mathbf{g}_\theta}(\mathbf{x})\right|,$$

where $J_{\mathbf{g}_\theta}(\mathbf{x}) = \frac{\partial \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \mathbf{x}}$ denotes the Jacobian of $\mathbf{g}_{\boldsymbol{\theta}}$ at $\mathbf{x}$. Given invertible transformations $\mathbf{g}_i : \mathbb{R}^d \to \mathbb{R}^d$ with parameter $\boldsymbol{\theta}_i$, $i \in [K]$, they can be composed together to construct a more expressive transformation $\mathbf{g}_{\boldsymbol{\theta}} = \mathbf{g}_1 \circ \cdots \circ \mathbf{g}_K$, where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$. As $\mathbf{g}_{\boldsymbol{\theta}}$ is the composition of invertible mappings $\mathbf{g}_i$, it is invertible and its log density is given by $\log p(\mathbf{x}; \boldsymbol{\theta}) = \log \pi(\mathbf{g}_\theta(\mathbf{x})) + \sum_{i=1}^K \log\left|\det J_{\mathbf{g}_i}(\mathbf{h}_i; \boldsymbol{\theta}_i)\right|$, where $\mathbf{h}_i = \mathbf{g}_i(\mathbf{h}_{i-1}; \boldsymbol{\theta}_i)$ and $\mathbf{h}_0 = \mathbf{x}$.

Let $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a dataset consisting of i.i.d. observations, where $\mathbf{x}_i \in \mathbb{R}^d$. The parameters $\boldsymbol{\theta}$ of normalizing flows are typically learned via MLE, which requires computing the Jacobian determinant sub-transformations $\mathbf{g}_j$.

$$\arg\max_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \log \pi(\mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_i)) + \sum_{j=1}^{K} \log\left|\det J_{\mathbf{g}_j}(\mathbf{x}_i; \boldsymbol{\theta}_j)\right|$$

DP-HFlow (Heterogeneous Flow)

There are three main components in DP-HFlow: (i) a dequantization layer $\mathbf{g}_{\text{deq}} : \mathbb{R}^{d_c} \to \mathbb{R}^{d_c}$ to convert discrete categorical variables into continuous ones, (ii) a linear flow layer $\mathbf{g}_{\text{lin}} : \mathbb{R}^d \to \mathbb{R}^d$ to learn the correlation between variables, and (iii) an elementwise spline transformation $\mathbf{g}_{\text{spl}} : \mathbb{R}^d \to \mathbb{R}^d$ to model complex, multimodal densities.

Heterogeneous Data We start by defining heterogeneous data and fixing notations. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a dataset consisting of n observations. Each observation $\mathbf{x} = (x[1], x[2], \ldots, x[d])$ is a vector corresponding to d different measurements, where $x[j]$ denotes the j^{th} element of $\mathbf{x}$. We use $\mathbf{x}[k : \ell]$ to denote the entries with indices from k to ℓ. Each attribute value $x[j]$ can be either numerical or categorical. Specifically, without loss of generality, we assume that the first d_n attributes $x[1], \ldots, x[d_n]$ are numerical and the remaining $d_c = d - d_n$ attributes $\mathbf{x}[d_n{+}1 : d]$ are categorical. A numerical variable can takes values from a continuous domain (e.g., $\mathbb{R}$) or discrete integer values representing counts. A categorical variable $x[j]$ takes a value from a discrete set $\mathcal{X}_j$ and can either be nominal or ordinal. The number of categories, denoted by $|\mathcal{X}_j|$, varies between different variables.

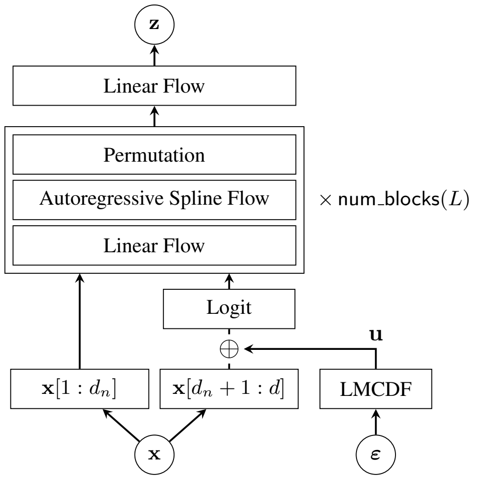

Figure 1: Architecture of DP-HFlow

Architecture

Variational Dequantization DP-HFlow dequantizes the discrete variables into continuous ones. Specifically, let $\mathbf{x}^{(c)} = \mathbf{x}[d_n{+}1 : d]$ denote the subset of $\mathbf{x}$ containing only categorical variables and $\mathbf{u} \in \mathbb{R}^{d_c}$ be a noise variable drawn from some probability distribution $\mathcal{P}$. The dequantized (continuous) data is obtained by adding real-valued noise to the discrete data: $\mathbf{y}^{(c)} = \mathbf{x}^{(c)} + \mathbf{u}$. We consider two ways to set the distribution $\mathcal{P}$: uniform dequantization (Theis, van den Oord, and Bethge 2016) and variational dequantization (Ho et al. 2019). In uniform dequantization, $\mathcal{P}$ is set to be a uniform distribution over $[0, 1]^{d_c}$ independently of data. On the other hand, the variational dequantization defines $\mathcal{P}$ to be a variational posterior distribution $q(\mathbf{u}|\mathbf{x}^{(c)})$. To build a flexible posterior, we choose $q(\mathbf{u}|\mathbf{x}^{(c)})$ to be a conditional normalizing flow with coupling transformation as in (Ho et al. 2019). Specifically, a coupling transformation is constructed using the cumulative density function (CDF) of mixture of M logistic distributions:

$$F_{\text{LMCDF}}(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}) = \sum_{i=1}^{M} \pi_i \sigma((x - \mu_i) \cdot \exp(-s_i)),$$

where $\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s} \in \mathbb{R}^M$ are the parameters of logistic mixture distribution corresponding to mixture weight, component means, and component scales, respectively, and $\sigma(\cdot)$ denotes the sigmoid function. Let $\boldsymbol{\varepsilon}$ be a noise vector drawn from a base distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_{d_c})$ and $\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2)$ be a partition of $\boldsymbol{\varepsilon}$ such that $\boldsymbol{\varepsilon}_1 \in \mathbb{R}^r$, $\boldsymbol{\varepsilon}_2 \in \mathbb{R}^{d_c - r}$, and $0 < r < d_c$. The dequantization transformation $\mathbf{g}_{\text{deq}}$ first draws a sample $\mathbf{u}$

from the posterior $q(\mathbf{u}|\mathbf{x}^{(c)})$ using the following conditional normalizing flow.

$$\mathbf{r} = \mathrm{NN}_\phi(\mathbf{x}^{(c)}), \quad \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s} = \mathrm{NN}_\psi([\boldsymbol{\varepsilon}_1, \mathbf{r}]),$$
$$\mathbf{u}_1 = \boldsymbol{\varepsilon}_1, \ \mathbf{u}_2 = F_{\mathrm{LMCDF}}(\boldsymbol{\varepsilon}_2; \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}), \ \mathbf{u} = \sigma([\mathbf{u}_1, \mathbf{u}_2]),$$

where $\mathrm{NN}_{\boldsymbol{\psi}}$ and $\mathrm{NN}_{\boldsymbol{\phi}}$ are the neural networks with parameters $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$, respectively, that output the parameters of logistic mixture distribution.

Linear Flows DP-HFlow uses elementwise transformations to deal with complex distributions. While being highly expressive, elementwise transformations are limited in their ability to capture all the interactions between variables as the transformation is applied dimensionwise rather than across dimensions. To help DP-HFlow capture the interactions between variables, we add linear transformations of the form

$$\mathbf{z} = \mathbf{g}_{\mathrm{lin}}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b},$$

where $\mathbf{W} \in \mathbb{R}^{d\times d}$ and $\mathbf{b} \in \mathbb{R}^d$ are weight and bias, respectively. However, the computation of Jacobian determinant of $\mathbf{g}_{\mathrm{lin}}$ (i.e., $\det(\mathbf{W})$) takes $\mathcal{O}(d^3)$ time, and hence existing approaches decompose $\mathbf{W}$ into a form that allows easy computation of Jacobian determinant, trading-off expressiveness and computational cost. We decompose $\mathbf{W}$ as the sum of diagonal matrix and low rank matrix (Lu and Huang 2020):

$$\mathbf{g}_{\mathrm{lin}}(\mathbf{x}) = (\mathrm{diag}(\mathbf{s}) + \mathbf{A}\mathbf{B})\mathbf{x} + \mathbf{b}, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{d\times r}$, $\mathbf{B} \in \mathbb{R}^{r\times d}$, $r > 0$ is the rank of $\mathbf{A}$ and $\mathbf{B}$. Using the matrix determinant lemma, we see that $|\det(\mathbf{W})| = |\det(\mathrm{diag}(\mathbf{s}))\cdot\det(\mathbf{I}_r + \mathbf{B}\,\mathrm{diag}(\mathbf{s})^{-1}\mathbf{A})|$. The second determinant needs to be numerically computed but we set r to a small number. [1] Another transformation used in DP-HFlow is based on LU-decomposition (Kingma and Dhariwal 2018):

$$\mathbf{g}_{\mathrm{lin}}(\mathbf{x}) = \mathbf{PLU}\mathbf{x} + \mathbf{b}, \tag{2}$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is a lower triangular matrix with ones on the diagonal and $\mathbf{U}$ is upper triangular matrix with non-zero diagonal entries. Notice that the above decomposition yields a simple Jacobian determinant $|\det\mathbf{W}| = |\prod_{i=1}^d U[i,i]|$. The permutation matrix $\mathbf{P}$ is randomly initialized and then fixed throughout the training (and hence it doesn't have any learnable parameters).

Autoregressive Spline Transformation To learn the complex joint distribution of $\mathbf{x}$, we construct an autoregressive elementwise transformation using a monotonic rational quadratic spline (Durkan et al. 2019). Let $\{(x^{(k)}, y^{(k)})\}_{k=0}^K$ be a set of monotonically increasing points, called knots, that satisfies $x^{(k-1)} < x^{(k)}$, $y^{(k-1)} < y^{(k)}$, for $k = 1, \ldots, K$, $x^{(0)} = y^{(0)} = -B$, and $x^{(K)} = y^{(K)} = B$, where $B > 0$ is a constant corresponding to the boundary of spline. Further, define $\Delta^{(k)} = (y^{(k+1)} - y^{(k)})/(x^{(k+1)} - x^{(k)})$ and $\xi(x) = (x - x^{(k)})/(x^{(k+1)} - x^{(k)})$. A rational quadratic spline is a piecewise polynomial function in which points x in the interval $[x^{(k)}, x^{(k+1)}]$, called a bin, are mapped to points in $[y^{(k)}, y^{(k+1)}]$ by the function

$$S_k(x) = y^{(k)} + \frac{(y^{(k+1)} - y^{(k)})[\Delta^{(k)}\xi^2 + \delta^{(k)}\xi(1-\xi)]}{\Delta^{(k)} + [\delta^{(k+1)} + \delta^{(k)} - 2\Delta^{(k)}]\xi(1-\xi)},$$

where $\delta^{(k)}$ is the derivatives at the kth knot. Notice that the spline for dimension j is fully specified by parameter vector $\boldsymbol{\theta}_j = [\boldsymbol{\theta}_j^{\mathrm{w}}, \boldsymbol{\theta}_j^{\mathrm{h}}, \boldsymbol{\theta}_j^{\mathrm{d}}]$, where $\theta_j^{\mathrm{w}}[k] = x^{(k)} - x^{(k-1)}$ and $\theta_j^{\mathrm{h}}[k] = y^{(k)} - y^{(k-1)}$ correspond to the width and height of the kth bin, respectively, and $\boldsymbol{\theta}^{\mathrm{d}} \in \mathbb{R}^{K-1}$ represents the derivatives at $K-1$ internal knots.

We parameterize the spline using a masked multilayer perceptron (MLP) following the approach used in MADE (Germain et al. 2015). Given an input vector $\mathbf{x} \in \mathbb{R}^d$, MADE assumes an ordering between variables $x[1] \prec x[2] \prec \ldots \prec x[d]$ and efficiently imposes autoregressive dependency using a binary mask. Let $\mathrm{NN}_{\boldsymbol{\nu}}$ denote the MADE network with parameter $\boldsymbol{\nu}$. Then the parameters that define the spline transformation are given by $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_d] = \mathrm{NN}_\nu(x[1:d])$. The binary masks in MADE ensure that $\boldsymbol{\theta}_i$ is only dependent on the input $x[1:i-1]$. Combining all together, the autoregressive transform $\mathbf{g}_{\mathrm{spl}}$ is defined by

$$\mathbf{g}_{\mathrm{spl}}(\mathbf{x}) = [S_1(x[1]; \boldsymbol{\theta}_1), \ldots, S_d(x[d]; \boldsymbol{\theta}_d)],$$

where $\boldsymbol{\theta}_k = \mathrm{NN}_{\nu_k}(\mathbf{x})$ is the parameters of the kth elementwise spline function S_k, generated by the MADE network.

Differentially Private Training

We now describe the innovative techniques proposed to improve the private training of normalizing flow model.

Conditional Spline Flow To reduce the parameter complexity, we modify the MADE network $\mathrm{NN}_{\boldsymbol{\nu}}$ to take an additional input $\mathbf{q}$ and to generate the output conditioned on $\mathbf{q}$. Let $\boldsymbol{\theta}^j$ denote the parameters of the spline transformation in the jth block of DP-HFlow. We have

$$\boldsymbol{\theta}^j = (\boldsymbol{\theta}_1^j, \ldots, \boldsymbol{\theta}_d^j) = \mathrm{NN}_{\boldsymbol{\nu}}(\mathbf{x}, \mathbf{q}(j)),$$

where $\mathbf{q}$ is the transformation of one-hot encoding of block index j, i.e., $\mathbf{q}(j)$ is an embedding of block index. This enables the sharing of parameter network $\mathrm{NN}_{\boldsymbol{\nu}}$ across blocks and greatly reduces the number of parameters to train while allowing to improve the expressiveness of model.

Per-layer Clipping Let $M_{\boldsymbol{\theta}}$ be a normalizing flow model consisting of multiple neural networks f^i, $i = 1, \ldots, J$, and $\boldsymbol{\theta}_\ell^i \in \mathbb{R}^H$ be the parameters of ℓth layer in f^i, $\ell = 1, \ldots, L$. For ease of illustration, we assume each network has L layers. All the parameters of $M_{\boldsymbol{\theta}}$ can be written as a vector $\boldsymbol{\theta} = (\boldsymbol{\theta}_1^1, \ldots, \boldsymbol{\theta}_L^J) \in \mathbb{R}^{JLH}$ by concatenating flattened parameters of each layer in networks f^i (either row-wise or column-wise). Let $\mathbf{g} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(M_{\boldsymbol{\theta}}(\mathbf{x})) = (\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}_1^1}, \ldots, \frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}_L^J})$ be the gradient of loss function $\mathcal{L}$ w.r.t. $\boldsymbol{\theta}$ for an example $\mathbf{x}$ in minibatch. The sensitivity of $\mathbf{g}$ can be bounded using the gradient clipping technique (Abadi et al. 2016). Given a threshold $C > 0$, the clipped gradient $\overline{\mathbf{g}}$ is obtained by

$$\overline{\mathbf{g}} = \mathrm{clip}_C(\mathbf{g}) = \min(1, C/\|\mathbf{g}\|_2)\cdot\mathbf{g}.$$

[1] In our experiments, we fix $r = 1$, which yields a closed form solution: $|(\prod_{i=1}^d s[i])(1 + \mathbf{BA} \oslash \mathbf{s})|$, where $\oslash$ denotes elementwise division.

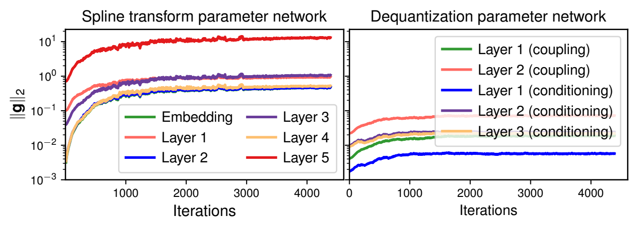

Figure 2: The scales of per-example gradients for 3 different neural networks used in DP-HFlow. The y-axis is in logscale.

In essence, it modifies $\mathbf{g}$ by multiplying the *same* constant $C/\|\mathbf{g}\|_2$ to all entries if its L_2 norm is greater than C. This is known as flat clipping. Figure 2 shows that the magnitude of per-example gradient varies across different networks, as well as across different layers within a network. When the networks f^i are heterogeneous (in terms of gradient scales), clipping $\mathbf{g}$ with the same threshold can have adverse impact, as it can introduce significantly larger bias into some layers. Alternatively, we can clip the gradient for each layer separately, called per-layer clipping (McMahan et al. 2018). DP-HFlow splits the global clipping threshold C between layers in such a way as to ensure that $\|\mathbf{g}\|_2 < C$. This is done by setting local thresholds to satisfy $C = \sqrt{\sum_{i=1}^{J}\sum_{\ell=1}^{L}(C_\ell^i)^2}$ and clipping $\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}_\ell^i}$ using a local threshold C_ℓ^i. In our implementation, the global clipping threshold is first split between networks and then between layers within a network proportional to their parameter complexity.

Fine-grained Gradient Clipping Prior work has shown that deep neural networks can be trained with sparsified gradients (Sun et al. 2017; Frankle and Carbin 2019; Raihan and Aamodt 2020). This implies that, in over-parameterized models, not all parameters equally contribute to the model accuracy and some units are more important for the accuracy than others. Inspired by this observation, we propose two advanced clipping methods that aim to further fine-tune the clipping threshold assignment within a layer. The goal is to reduce the extent to which the gradients that are important for good performance are clipped. Specifically, the proposed clipping methods identify the important units using the magnitude of gradients as a criterion, and assign larger clipping threshold values to them. This has an effect of increasing the magnitude of gradients for the units, and hence they are likely to become more noise resistant. Consider a linear layer ℓ with m units. Let $\mathbf{W} \in \mathbb{R}^{m\times n}$ be its weight matrix and $\mathbf{G}$ denote the (per-example) gradient w.r.t. $\mathbf{W}$, i.e., $\mathbf{G} = \nabla_{\mathbf{W}}\mathcal{L}$.

- **Per-unit clipping** Given a clipping threshold C_ℓ for the layer, the threshold for the ith unit $C_\ell[i]$ is set as

$$C_\ell[i] = C_\ell\sqrt{\|G[i,:]\|_1 / \sum_{j=1}^{m}\|G[j,:]\|_1}, \text{ for } i = 1,\ldots,m\,,$$

where $G[i,:]$ denotes the ith row of $\mathbf{G}$. This allows the entries with larger magnitude to get clipped less than they would in a naive clipping. Given C_ℓ, the clipping is applied to each unit:

$$\overline{\mathbf{g}}_\ell = \left(\mathrm{clip}_{C_\ell[1]}(G[1,:]),\ldots,\mathrm{clip}_{C_\ell[m]}(G[m,:])\right)\,.$$

- **Stochastic sparsification** Let $\mathbf{g}_\ell = \mathrm{vec}(\mathbf{G}) \in \mathbb{R}^{mn}$ be the vectorized gradient for layer ℓ and τ be the kth largest entry of $\mathbf{g}_\ell$ in absolute value. The stochastic sparsification randomly sets the entries of $\mathbf{g}_\ell$ whose magnitude is smaller than τ to zero using the following operator (Ye et al. 2020):

$$T_{\tau,\gamma}(x) = \begin{cases} x & \text{if } |x| > \tau, \\ \mathrm{sign}(x)\cdot\tau & \text{if } \tau\cdot\gamma \le |x| \le \tau, \\ 0 & \text{if } |x| < \tau\cdot\gamma, \end{cases}$$

where $\gamma \sim \mathcal{U}[0,1]$ is a uniform random variable. It is easy to see that $T_{\tau,\gamma}$ is unbiased, i.e., $\mathbb{E}[T_{\tau,\gamma}(X)] = \mathbb{E}[X]$. The clipping is applied to the sparsified gradient, i.e., $\overline{\mathbf{g}}_\ell = \mathrm{clip}_{C_\ell}(T_{\tau,\gamma}(\mathbf{g}_\ell))$.

Both strategies share the same goal of assigning larger clipping threshold to gradients with larger absolute value so that important learning signals can get less clipped.

Fast Per-example Gradient Computation To efficiently compute the per-example gradients, we extend the fast gradient clipping framework of (Lee and Kifer 2021) to those layers with non-standard parameterizations, for example, the linear transformation in (1) and (2). Here we only report the final equation for per-example gradients of these linear transformations, and the full derivation is presented in the supplementary document. The gradients w.r.t. the parameters of LU decomposition are given by

$$\frac{\partial\mathcal{L}}{\partial\mathbf{L}} = \frac{\partial\mathcal{L}}{\partial\mathbf{z}}\cdot(\mathbf{U}\mathbf{x})^\intercal\,,\quad \frac{\partial\mathcal{L}}{\partial\mathbf{U}} = \mathbf{L}^\intercal\frac{\partial\mathcal{L}}{\partial\mathbf{z}}\mathbf{x}^\intercal\,, \tag{3}$$

and those for the low-rank decomposition are

$$\frac{\partial\mathcal{L}}{\partial\mathbf{A}} = \frac{\partial\mathcal{L}}{\partial\mathbf{z}}(\mathbf{B}\mathbf{x})^\intercal\,,\ \frac{\partial\mathcal{L}}{\partial\mathbf{B}} = \mathbf{A}^\intercal\frac{\partial\mathcal{L}}{\partial\mathbf{z}}\mathbf{x}^\intercal\,,\ \frac{\partial\mathcal{L}}{\partial\mathbf{s}} = \frac{\partial\mathcal{L}}{\partial\mathbf{z}}\odot\mathbf{x}\,, \tag{4}$$

where $\mathcal{L}$ denotes the objective function of DP-HFlow and $\odot$ denotes elementwise product. For both transformations, the computation of per-example gradients involve two quantities, the $\frac{\partial\mathcal{L}}{\partial\mathbf{z}}$ (the gradient of loss function w.r.t. the output of transformation) and intermediate computation results (e.g., $\mathbf{U}\mathbf{x}$ in (3) and $\mathbf{B}\mathbf{x}$ in (4)). During the backward phase, the backpropagation algorithm computes $\frac{\partial\mathcal{L}}{\partial\mathbf{z}}$ and passes it through layers. Our implementation stores intermediate computation results during the feed-forward phase and computes the per-example gradients in the backward phase according to Equation (3) and (4).

Experimental Results

To evaluate the performance of DP-HFlow, we perform experiments on four real datasets: Adult, Census, Covertype, Intrusion. Adult and Census are extracted from the US Census and contain demographic and employment-related information. Covertype is a dataset for predicting forest cover

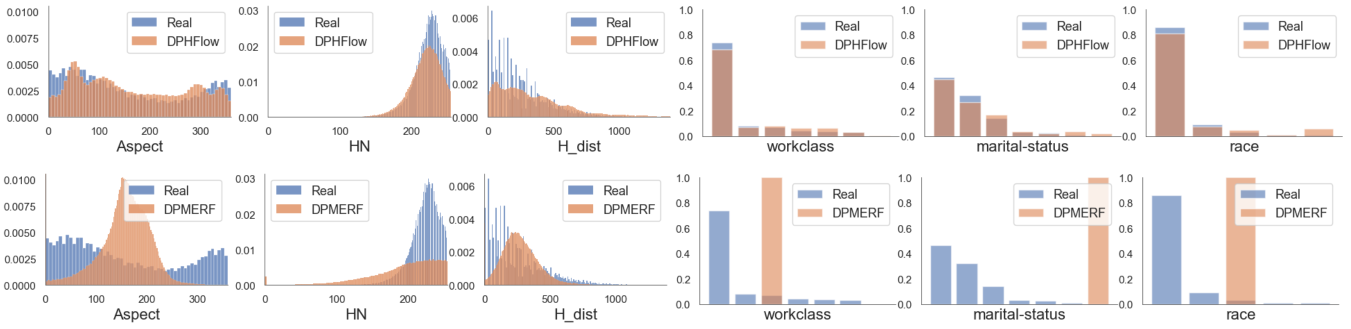

Figure 3: The marginal distribution of data generated by DP-HFlow (upper panels) and DP-MERF (lower panels) for Covertype (left panels) and Adult (right pannels) under $(1, 10^{-5})$-DP

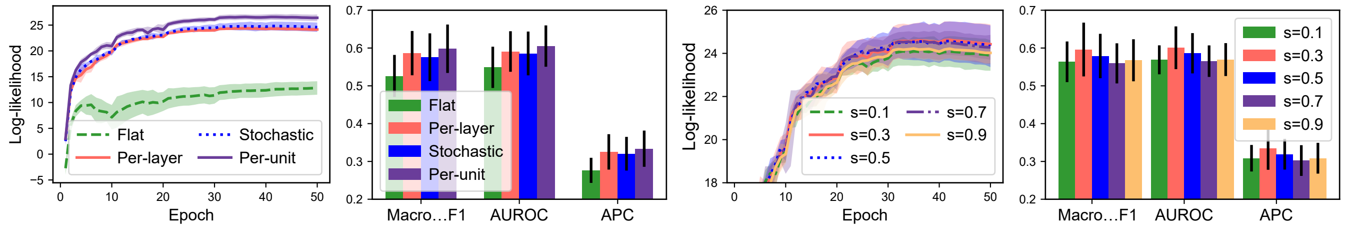

Figure 4: Log-likelihood and classification performance by varying clipping methods (left) and by varying sparsity level (right)

types using cartographic variables such as elevation, aspect, and slope. Intrusion dataset contains attributes collected from network traffic simulations. The quality of generated samples is assessed using three criteria: (i) the ability to match the marginal distribution of original dataset, (ii) the ability to preserve dependency structure in the original dataset and (iii) the test performance of classifiers trained on the synthetic dataset. The performance of DP-HFlow is compared with those of two state-of-the-art methods, DP-MERF and GS-WGAN and that of DP-CGAN (Torkzadehmahani, Kairouz, and Paten 2019). We also use CTGAN (Xu et al. 2019) as our baseline for non-private setting. All experiments were performed on a server with an NVIDIA RTX 8000 GPU. In all experiments, DP-HFlow is instantiated by stacking 3 blocks of autoregressive spline transformation and low rank-based linear transformation on top of dequantization layer. A reverse ordering permutations is inserted in between blocks. We used AdaBelief optimizer (Zhuang et al. 2020) with learning rate 0.001 and default smoothing parameter of $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Marginal Distribution Comparison We investigate the ability of each generative model to learn the marginal distribution of an attribute. Figure 3 shows the marginal distribution of samples. The histograms in blue represent the distribution of original dataset. The distribution of generated samples in orange color is overlaid on top of the original distribution. The left three columns in Figure 3 correspond to 3 continuous variables selected from Covertype dataset (the full set of graphs is provided in the supplementary material). The marginal distribution of Aspect attribute shows that DP-HFlow can learn the distribution with two modes while DP-MERF fails to capture them and collapses onto their average. We also observed that DP-MERF tends to generate symmetrically distributed samples even when the original data is highly skewed and that it often underestimates the variance of attribute. This is shown in the graph in the third column of Figure 3. The right three columns shows the marginal distributions of 3 discrete variables selected from Adult dataset. As it was the case for continuous variables, there is no diversity at all in the sample generated by DP-MERF as it wasn't specifically designed to handle discrete variables. The results show that DP-HFlow can capture the complex marginal distribution. In contrast, both baselines largely fail to match the distribution and suffer from the mode collapse issue, rendering their outputs inappropriate for exploratory data analysis (EDA) purpose.

	DPCGAN	GSWGAN	DPMERF	**DPHFlow**
RMSE	0.4434	0.2058	0.1863	**0.0717**
MAE	0.3549	0.1396	0.1342	**0.0482**

Table 1: RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) between Kendall's τ matrices for the Covertype dataset under $(1, 10^{-5})$-DP setting.

Dependency Structure Comparison We now evaluate DP-HFlow's ability to capture the dependency between variables. We compute pairwise Kendall's τ on the Covertype dataset to see if dependencies between continuous variables in the original data are preserved in the synthetic data. Table 1 presents the distance between Kendall's τ matrices of original and synthetic datasets. As shown, DP-HFlow preserves the dependency structure better than other methods. We obtained similar results on other datasets (the detailed results are provided in the supplementary material).

		Real	Private				Non-Private		
			DP-CGAN	GS-WGAN	DP-MERF	**DP-HFlow**	CTGAN	DP-MERF	**DP-HFlow**
Macro-F1	Adult	0.79±0.02	0.46±0.07	0.42±0.09	0.37±0.15	**0.56±0.07**	0.74±0.02	0.41±0.16	**0.75±0.01**
	Census	0.75±0.01	0.45±0.09	0.44±0.13	0.48±0.14	**0.52±0.03**	0.67±0.04	0.50±0.14	**0.70±0.04**
	Covertype	0.77±0.16	0.15±0.03	0.11±0.03	**0.31±0.05**	0.22±0.03	0.22±0.04	0.29±0.05	**0.49±0.04**
	Intrusion	0.86±0.07	0.19±0.09	0.13±0.08	0.36±0.05	**0.40±0.03**	**0.54±0.05**	0.38±0.06	0.46±0.06

Table 2: Performance comparison of different methods using macro F1-score metric under the private and non-private setting.

		Real	Private				Non-Private		
			DP-CGAN	GS-WGAN	DP-MERF	**DP-HFlow**	CTGAN	DP-MERF	**DP-HFlow**
AUROC	Adult	0.90±0.02	0.53±0.14	0.48±0.11	0.63±0.09	**0.75±0.05**	0.86 ±0.02	0.65±0.10	**0.87±0.02**
	Census	0.93±0.02	0.50±0.16	0.56±0.20	0.68±0.13	**0.78±0.06**	0.89±0.04	0.69±0.11	**0.91±0.04**
APC	Adult	0.77±0.04	0.28±0.09	0.25±0.05	0.34±0.08	**0.50±0.07**	0.66±0.04	0.39±0.10	**0.70±0.05**
	Census	0.59±0.06	0.07±0.03	0.10±0.06	0.15±0.07	**0.17±0.05**	0.43±0.07	0.15±0.06	**0.49±0.08**

Table 3: Performance comparison of different methods using AUROC and APC metric under the private and non-private setting.

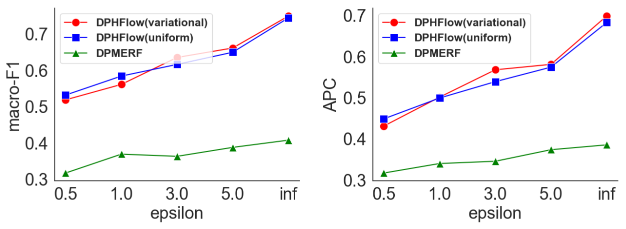

Figure 5: The effect of different dequantization methods on the classification performance by varying privacy budget for the Adult dataset.

Classification Performance We assess the quality of generated samples in the context of predictive modeling. For this experiment, we train 9 classifiers including logistic regression, decision tree, and gradient boosting (the full details can be found in the supplementary material) on the synthetic dataset generated by the models. The performance of these classifiers are evaluated on the original dataset spared for testing. In other words, we test whether the predictive models trained on synthetic datasets can generalize to the real datasets. We repeat each experiment 10 times and report averaged values. Table 2 and 3 report macro F1-score, AUROC (area under the receiver operating characteristics curve), and APC (average precision score) averaged over all classifiers. The performance metrics for individual classifier are provided in the supplementary material. DP-HFlow outperforms other baselines on Adult, Census and Intrusion datasets. Interestingly, we found that DP-MERF outperforms other methods on Covertype dataset although the marginal distribution of synthetic data generated by the method largely deviates from that of original dataset.

Effect of Parameters The left two graphs in Figure 4 demonstrate the performance of DP-HFlow under 4 different clipping methods. These set of experiments were performed using the Adult dataset. The result shows that there exists a large gap in the log-likelihood between flat clipping and other three methods. Yet, their classification performance yields little difference. The result implies that find-grained clipping methods help avoid excessively cutting important learning signals in the gradients. The right two graphs shows the impact of varying sparsity level on the performance of DP-HFlow trained with stochastically sparsified gradients. Let $\mathbf{g} \in \mathbb{R}^m$ be the gradient. Given the the sparsity level s, the threshold value τ in stochastic sparsification is chosen as the k^{th} largest entry in absolute value where $k = m(1 - s)$. Notice that it reduces to per-layer clipping when $s = 0$. As shown in the figure, the performance of DP-HFlow is not sensitive to the choice of the sparsity level and the moderate levels of sparsity lead to similar performance. Figure 5 describes the impact of different dequantization methods, uniform and variational, on the classification performance. We observed that, in low privacy regime (i.e., large ϵ), samples generated with variational dequantization led to better classification performance. When ϵ is small, the uniform dequantization showed better performance. This is because the model can allocate larger privacy budget to the density estimator (i.e., spline flow) as the uniform dequantization does not require training an extra neural network.

Conclusion

We presented a Rényi differentially private normalizing flow model for synthesizing tabular datasets. The fine-grained gradient clipping methods proposed to train parameter-heavy NF models are simple but shown to be highly effective in accelerating private training. In this work, we used the magnitude of gradients as a criterion to determine the importance of parameters. Finding other criteria is an interesting direction to explore and left as a future work. We believe that the methods can help boost the performance of other differentially private models.

Acknowledgements

This research was supported by the National Science Foundation under Grant No. 1943046.

References

Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H. B.; Mironov, I.; Talwar, K.; and Zhang, L. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 308–318. ACM.

Bagdasaryan, E.; Poursaeed, O.; and Shmatikov, V. 2019. Differential Privacy Has Disparate Impact on Model Accuracy. *Advances in Neural Information Processing Systems*, 32: 15479–15488.

Balog, M.; Tolstikhin, I.; and Schölkopf, B. 2018. Differentially Private Database Release via Kernel Mean Embeddings. In *International Conference on Machine Learning*, 414–422.

Chen, D.; Orekondy, T.; and Fritz, M. 2020. GS-WGAN: A Gradient-Sanitized Approach for Learning Differentially Private Generators. *Advances in Neural Information Processing Systems*, 33: 12673–12684.

Chen, Q.; Xiang, C.; Xue, M.; Li, B.; Borisov, N.; Kaafar, D.; and Zhu, H. 2018. Differentially Private Data Generative Models. *CoRR*, abs/1812.02274.

Chu, C.; Minami, K.; and Fukumizu, K. 2020. Smoothness and Stability in GANs. In *International Conference on Learning Representations*.

Durkan, C.; Bekasov, A.; Murray, I.; and Papamakarios, G. 2019. Neural Spline Flows. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; dAlché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Dwork, C.; Kenthapadi, K.; McSherry, F.; Mironov, I.; and Naor, M. 2006a. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 486–503. Springer.

Dwork, C.; McSherry, F.; Nissim, K.; and Smith, A. 2006b. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, 265–284. Springer.

Frankle, J.; and Carbin, M. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*.

Germain, M.; Gregor, K.; Murray, I.; and Larochelle, H. 2015. MADE: Masked Autoencoder for Distribution Estimation. In *International Conference on Machine Learning*, 881–889. PMLR.

Harder, F.; Adamczewski, K.; and Park, M. 2021. DP-MERF: Differentially Private Mean Embeddings with RandomFeatures for Practical Privacy-Preserving Data Generation. In *International Conference on Artificial Intelligence and Statistics*, 1819–1827. PMLR.

Ho, J.; Chen, X.; Srinivas, A.; Duan, Y.; and Abbeel, P. 2019. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design. In *International Conference on Machine Learning*, 2722–2730. PMLR.

Jordon, J.; Yoon, J.; and van der Schaar, M. 2019. PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees. In *ICLR*.

Kingma, D. P.; and Dhariwal, P. 2018. Glow: Generative Flow with Invertible 1x1 Convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 10236–10245. Red Hook, NY, USA: Curran Associates Inc.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In Bengio, Y.; and LeCun, Y., eds., *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Lee, J.; and Kifer, D. 2021. Scaling up Differentially Private Deep Learning with Fast Per-Example Gradient Clipping. *Proceedings on Privacy Enhancing Technologies*, 2021(1): 128–144.

Lu, Y.; and Huang, B. 2020. Woodbury transformations for deep generative flows. *Advances in Neural Information Processing Systems*, 33.

McMahan, H. B.; Ramage, D.; Talwar, K.; and Zhang, L. 2018. Learning Differentially Private Recurrent Language Models. In *International Conference on Learning Representations*.

Mironov, I. 2017. Renyi differential privacy. In *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*, 263–275. IEEE.

Papernot, N.; Abadi, M.; Erlingsson, Ú.; Goodfellow, I.; and Talwar, K. 2017. Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data. In *Proceedings of the International Conference on Learning Representations*.

Raihan, M. A.; and Aamodt, T. 2020. Sparse Weight Activation Training. *Advances in Neural Information Processing Systems*, 33: 15625–15638.

Rezende, D.; and Mohamed, S. 2015. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning*, 1530–1538. PMLR.

Sun, X.; Ren, X.; Ma, S.; and Wang, H. 2017. MeProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 3299–3308. Sydney, NSW, Australia: JMLR.org.

Tabak, E. G.; and Turner, C. V. 2013. A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics*, 66(2): 145–164.

Tantipongpipat, U.; Waites, C.; Boob, D.; Siva, A.; and Cummings, R. 2020. Differentially Private Mixed-Type Data Generation For Unsupervised Learning.

Theis, L.; van den Oord, A.; and Bethge, M. 2016. A Note on the Evaluation of Generative Models. *arXiv:1511.01844 [cs, stat]*.

Torkzadehmahani, R.; Kairouz, P.; and Paten, B. 2019. DP-CGAN: Differentially Private Synthetic Data and Label

Generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

Uria, B.; Murray, I.; and Larochelle, H. 2013. RNADE: The Real-Valued Neural Autoregressive Density-Estimator. In Burges, C. J. C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Waites, C.; and Cummings, R. 2021. Differentially Private Normalizing Flows for Privacy-Preserving Density Estimation. *arXiv:2103.14068 [cs, stat]*.

Xie, L.; Lin, K.; Wang, S.; Wang, F.; and Zhou, J. 2018. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*.

Xu, L.; Skoularidou, M.; Cuesta-Infante, A.; and Veeramachaneni, K. 2019. Modeling Tabular data using Conditional GAN. *Advances in Neural Information Processing Systems*, 32: 7335–7345.

Ye, X.; Dai, P.; Luo, J.; Guo, X.; Qi, Y.; Yang, J.; and Chen, Y. 2020. Accelerating CNN Training by Pruning Activation Gradients. In Vedaldi, A.; Bischof, H.; Brox, T.; and Frahm, J.-M., eds., *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, 322–338. Cham: Springer International Publishing. ISBN 978-3-030-58595-2.

Zhuang, J.; Tang, T.; Ding, Y.; Tatikonda, S. C.; Dvornek, N.; Papademetris, X.; and Duncan, J. 2020. AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. *Advances in Neural Information Processing Systems*, 33.