Modeling Tabular data using Conditional GAN

Lei Xu LEIX@MIT.EDU
Laboratory for Information & Decision Systems
Massachusetts Institute of Technology
Cambridge, MA

Maria Skoularidou MS2407@CAM.AC.UK
MRC Biostatistics Unit
University of Cambridge
Cambridge, UK

Alfredo Cuesta-Infante ALFREDO.CUESTA@URJC.ES
Universidad Rey Juan Carlos
Móstoles, Spain

Kalyan Veeramachaneni KALYANV@MIT.EDU
Laboratory for Information & Decision Systems
Massachusetts Institute of Technology
Cambridge, MA

Editor:

Abstract

Modeling the probability distribution of rows in tabular data and generating realistic synthetic data is a non-trivial task. Tabular data usually contains a mix of discrete and continuous columns. Continuous columns may have multiple modes whereas discrete columns are sometimes imbalanced making the modeling difficult. Existing statistical and deep neural network models fail to properly model this type of data. We design `TGAN`, which uses a conditional generative adversarial network to address these challenges. To aid in a fair and thorough comparison, we design a benchmark with 7 simulated and 8 real datasets and several Bayesian network baselines. `TGAN` outperforms Bayesian methods on most of the real datasets whereas other deep learning methods could not.

1. Introduction

Recent developments in deep generative models have led to a wealth of possibilities. Using images and text, these models can learn the underlying probability distributions and generate high-quality samples. Over the past two years, the promise of such models has encouraged the development of generative adversarial networks (GANs) (Goodfellow et al., 2014) for tabular data generation. The idea of being GANs offers greater flexibility in modeling distributions than their statistical counterparts. This proliferation of new GANs brought up this question "Can these new GANs offer better generative models for their statistical counterpart?". To answer this question and evaluate these GANs, we used a group of real datasets to set-up a benchmarking system and implemented three of the most recent techniques [1]. For comparison purposes, we created two baseline methods using Bayesian networks. After testing these models using both simulated and real datasets, we found that modeling tabular data poses unique challenges for GANs, causing them to fall short of the baseline methods on a number of metrics, including the machine learning efficacy of the synthetically generated data. These challenges include the need to simultaneously model discrete and continuous columns, the multi-modality of information within each column, and the severe imbalance of categorical columns (we describe these challenges in detail in Section 3).

To address these challenges, in this paper, we propose `TGAN`, a method which introduces several new techniques including: augmenting training procedures with reversible data transforms, architectural changes to the neural networks, and addressing data imbalance by employing a novel conditional GAN (described in detail in section 4). When applied to the same datasets with the new benchmarking suite, `TGAN` performs significantly better than both the Bayesian network baselines and the other new GANs tested, as shown in Table 1. The contributions of this paper are as follows:

(1) Conditional GANs for synthetic data generation. We propose `TGAN` as a synthetic tabular data generator to address several of the issues mentioned above. `TGAN` outperforms all methods to date and surpasses Bayesian networks on at least 87.5% of our datasets. To further challenge `TGAN`, we adapted a variational autoencoder (VAE) (Kingma and Welling, 2013) for mixed-type tabular data generation. We call this `TVAE`. VAEs directly use data to build the generator; even with this advantage, we show that our proposed `TGAN` achieves competitive performance across many datasets and outperforms `TVAE` 3 times.

(2) A benchmarking system for synthetic data generation algorithms.[2] We designed a comprehensive benchmark framework using several tabular datasets and different evaluation metrics as well as implementations of several baselines and state-of-the-art methods. Our system is open source and can be extended with other methods and additional datasets. At the time of this writing, the benchmark has 5 deep learning methods, 2 Bayesian network methods, and 15 datasets.

2. Related Work

During the past decade, synthetic data has been generated by treating each column in a table as a random variable, modeling a joint multivariate probability distribution, and then sampling from that distribution. For example, a set of discrete variables may have been modeled using decision trees (Reiter, 2005) and Bayesian networks (Aviñó et al., 2018; Zhang et al., 2017). Spatial data could be modeled with a spatial decomposition tree (Cormode et al., 2012; Zhang et al., 2016). A set of non-linearly correlated continuous variables could be modeled using *copulas* (Patki et al., 2016; Sun et al., 2018). These models are restricted by the type of distributions and by computational issues, severely limiting the synthetic data's fidelity.

[1] We call our system SDGym as it evaluates generative modeling capability in terms of the model's ability to generate realistic synthetic data.

[2] Our benchmark can be found on `https://github.com/DAI-Lab/SDGym`.

Table 1: The number of wins of a particular method compared with the corresponding Bayesian network against an appropriate metric on 8 real datasets.

	# outperform	
Method	CLBN (Chow and Liu, 1968)	PrivBN (Zhang et al., 2017)
MedGAN (Choi et al., 2017)	1	1
VeeGAN (Srivastava et al., 2017)	0	2
TableGAN (Park et al., 2018)	3	3
TGAN	**7**	**8**

The development of generative models using VAEs and, subsequently, GANs and their numerous extensions (Arjovsky et al., 2017; Gulrajani et al., 2017; Zhu et al., 2017; Yu et al., 2017), has been very appealing due to the performance and flexibility offered in representing data. GANs are also used in generating tabular data, especially healthcare records; for example, (Yahi et al., 2017) uses GANs to generate continuous time-series medical records and (Camino et al., 2018) proposes the generation of discrete tabular data using GANs. medGAN (Choi et al., 2017) combines an auto-encoder and a GAN to generate heterogeneous non-time-series continuous and/or binary data. ehrGAN (Che et al., 2017) generates augmented medical records. tableGAN (Park et al., 2018) tries to solve the problem of generating synthetic data using a convolutional neural network which optimizes the label column's quality; thus, generated data can be used to train classifiers. PATE-GAN (Jordon et al., 2019) generates differentially private synthetic data.

3. Challenges with GANs in Tabular Data Generation Task

The task of synthetic data generation task requires training a data synthesizer G learnt from a table $\mathbf{T}$ and then using G to generate a synthetic table $\mathbf{T}_{syn}$. A table $\mathbf{T}$ contains N_c continuous columns $\{C_1, \ldots, C_{N_c}\}$ and N_d discrete columns $\{D_1, \ldots, D_{N_d}\}$, where each column is considered to be a variable. These random variables follow an unknown joint distribution $\mathbb{P}(C_{1:N_c}, D_{1:N_d})$. One row $\mathbf{r}_j = \{c_{1,j}, \ldots, c_{N_c,j}, d_{1,j}, \ldots, d_{N_d,j}\},\ j \in \{1, \ldots, n\}$, is one observation from the joint distribution. $\mathbf{T}$ is partitioned into training set $\mathbf{T}_{train}$ and test set $\mathbf{T}_{test}$. After training G on $\mathbf{T}_{train}$, $\mathbf{T}_{syn}$ is constructed by independently sampling rows from G. We evaluate the efficacy of a generator along 2 axes.

- **Likelihood fitness**: Columns in $\mathbf{T}_{syn}$ follow the same joint distribution as $\mathbf{T}_{train}$.
- **Machine learning efficacy**: We train a classifier or a regressor to predict one column using other columns as features. Such classifier or regressor learned from $\mathbf{T}_{syn}$ can achieve a similar performance on $\mathbf{T}_{test}$, as would a model learned on $\mathbf{T}_{train}$.

Several unique properties of tabular data challenge the design of a GAN model. In this section we highlight these challenges in increasing order of the complexity of solution required to solve them. In Table 2, we note which subset of these is addressed by the existing methods.

Table 2: A summary showing whether existing methods and our TGAN explicitly address these challenges [C1 - C5]. (* indicates it is able to model continuous and binary.)

Problems	**MedGAN**	**TableGAN**	**PATE-GAN**	**TGAN**
C1	✓*	✓*	✓*	✓
C2	x	x	x	✓
C3	x	✓	x	✓
C4	x	x	x	✓
C5	x	✓	x	✓

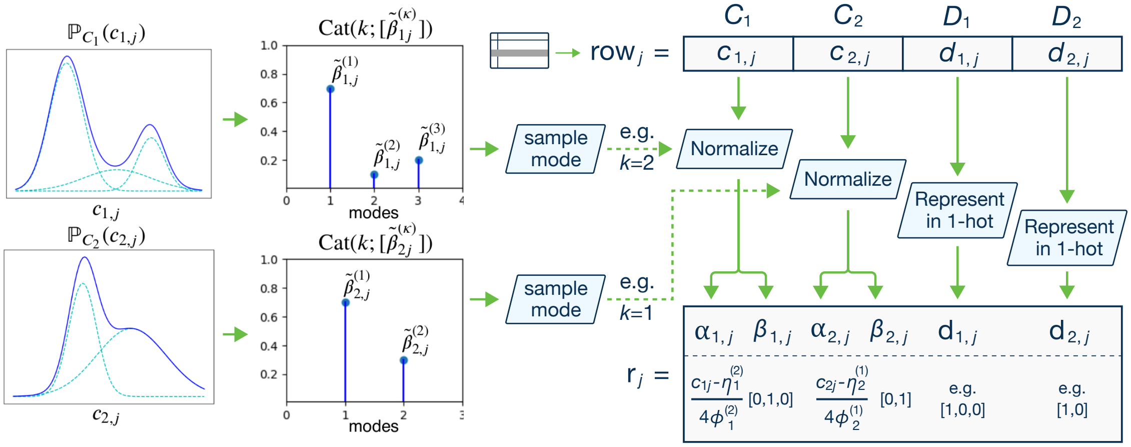
C1. Mixed data types. Real-world tabular data consists of mixed types (i.e. continuous, ordinal, categorical etc). To simultaneously generate a mix of discrete and continuous columns, modifications to GANs must apply both softmax and tanh on the output.

C2. Non-Gaussian distributions: In images, a pixel's values follow a Gaussian-like distribution, which can be normalized to $[-1, 1]$ using a min-max tranform. A tanh function is usually employed in the last layer of a network to output a value in this range. Continuous variables in tabular data are usually non-Gaussian and have distributions with long tails; thus most generated values will not be centred around zero. The gradient of tanh where most values will be located is flat - a phenomenon known as gradient saturation. This results in the model's inability to learn via gradients.

C3. Multimodal distributions. Continuous columns in tabular data usually have multiple modes. We observe that 57/123 continuous columns in our 8 real-world datasets have multiple modes. Srivastava et al. (2017) showed that vanilla GAN couldn't model all modes on a simple 2D dataset; thus it also wouldn't be able to model the multimodal distribution of continuous columns. To solve this problem and C2, we employ mode-specific pre-processing techniques as described in Section 4.1 and use PacGAN (Lin et al., 2018) to overcome mode collapse.

C4. Learning from sparse one-hot-encoded vectors. To enable learning from non-ordinal categorical columns, a categorical column is converted into a one-hot vector. When generating synthetic samples, a generative model is trained to generate a probability distribution over all categories using softmax. This is problematic in GANs because a trivial discriminator can simply distinguish real and fake data by checking the distribution's sparseness instead of considering the overall realness of a row. TGAN avoids such pathologies by applying gumbel-softmax (Jang et al., 2016) to generate a sparse and differentiable distribution over all categories.

C5. Highly imbalanced categorical columns. In real world datasets, most categorical columns have highly imbalanced distribution. In our datasets we noticed that 636/1048 of the categorical columns are highly imbalanced, in which the major category appears in more than 90% of the rows, resulting in severe mode collapse. Missing a minor category only causes tiny changes to the data distribution, but imbalanced data leads to insufficient training opportunities for minor classes. The critic network cannot detect such issue unless mode-collapse-preventing mechanisms such as PacGAN are used. These mechanisms can



$\mathbb{P}_{C_1}(c_{1,j})$
$c_{1,j}$
$\mathbb{P}_{C_2}(c_{2,j})$
$c_{2,j}$
$\mathrm{Cat}(k; [\tilde{\beta}_{1j}^{(\kappa)}])$
$\tilde{\beta}_{1,j}^{(1)}$
$\tilde{\beta}_{1,j}^{(2)}$
$\tilde{\beta}_{1,j}^{(3)}$
modes
$\mathrm{Cat}(k; [\tilde{\beta}_{2j}^{(\kappa)}])$
$\tilde{\beta}_{2,j}^{(1)}$
$\tilde{\beta}_{2,j}^{(2)}$
modes
sample mode
e.g. k=2
sample mode
e.g. k=1
$\mathrm{row}_j =$
C_1 C_2 D_1 D_2
$c_{1,j}$ $c_{2,j}$ $d_{1,j}$ $d_{2,j}$
Normalize
Normalize
Represent in 1-hot
Represent in 1-hot
$\mathrm{r}_j =$
$\alpha_{1,j}$ $\beta_{1,j}$ $\alpha_{2,j}$ $\beta_{2,j}$ $\mathrm{d}_{1,j}$ $\mathrm{d}_{2,j}$
$\frac{c_{1j}-\eta_1^{(2)}}{4\phi_1^{(2)}}$ [0,1,0] $\frac{c_{2j}-\eta_2^{(1)}}{4\phi_2^{(1)}}$ [0,1] e.g. [1,0,0] e.g. [1,0]

Figure 1: Reversible data transformation of a row with two continuous and two discrete columns. In this example we have assumed that C_1 consists of three gaussian components and C_2 consists of two; while $D_1 \in \{1, 2, 3\}$ and $D_2 \in \{1, 2\}$. Additionaly, we have assumed that the mode selected for C_1 was $k = 2$, the mode selected for C_2 was $k = 1$, the value $d_{1,j} = 1$ and the value $d_{2,j} = 1$. The resulting vector $\mathbf{r_j}$ has size $|\mathbf{r_j}| = 12$.

prevent GANs from generating only the most salient category. Synthetic data for minor categories are expected to be of lower quality, necessitating resampling.

4. TGAN Model

In this section, we explain our preprocessing method and introduce our `TGAN` model.
Notations: Besides the common operations like `tanh`, `ReLU`, `softmax`, batch normalization (Ioffe and Szegedy, 2015) as `BN` and dropout (Srivastava et al., 2014) as `drop`, we define

- $\mathrm{Cat}(x, [p_1, p_2, \ldots])$: categorical PMF of x with parameters $p_1, p_2, \ldots$
- $x_1 \oplus x_2 \oplus \ldots$: concatenate vectors $x_1, x_2, \ldots$
- $\mathtt{gumbel}_\tau(x)$: apply Gumbel softmax(Jang et al., 2016) with parameter τ on a vector x
- $\mathtt{leaky}_\gamma(x)$: apply a leaky ReLU activation on x with leaky ratio γ
- $\mathtt{FC}_{u \to v}(x)$: project linearly a u-dimensional vector x to a v-dimensional vector by means of a fully connected neural network with linear activation.

For readability, when we define our model, we replace the τ, γ, u, and v for `gumbel`, `leaky`, and `FC` with *actual* settings in the experiments. Additionally, we use $\mathbb{P}$ to stress that a given function is a probability distribution.

4.1 Reversible Data Transformations

In order to deal with mixed data types, each column is processed independently, according to whether its values are continuous or discrete. Figure 1 summarizes the transformation. A discrete column $d_{i,j}$ is simply transformed into a one-hot representation $\mathbf{d}_{i,j}$. For continuous columns, we use a *mode-specific* normalization, which is able to deal with non-Gaussian and multimodal distributions.

The mode-specific normalization consists of four steps. Let $c_{i,j}$ be a continuous value corresponding to the ith continuous column, C_i, and the jth row in the tabular data.

1. Begin by estimating the number of modes in the distribution of C_i. To do so, we use a variational Gaussian mixture model (VGM) (Bishop, 2006) that produces the probabilistic model $\mathbb{P}_{C_i}(c_{i,j})$, a Gaussian Mixture of m_i components with means $\eta_i^{(1)}, \ldots, \eta_i^{(m_i)}$, standard deviations $\phi_i^{(1)}, \ldots, \phi_i^{(m_i)}$ and weights $u^{(1)}, \ldots, u^{(m_i)}$ respectively,
$$\mathbb{P}_{C_i}(c_{i,j}) = \sum\nolimits_{\kappa=1}^{m_i} u^{(\kappa)} \mathcal{N}\left(c_{i,j} \ ; \ \eta_i^{(\kappa)}, \phi_i^{(\kappa)}\right)_.$$
The VGM model $\mathbb{P}_{C_i}(c_{i,j})$ is trained to maximize the likelihood on the training data.
2. Compute the PMF of the value $c_{i,j}$ sampled from each of the m_i modes as
$$\mathrm{Cat}(k; [\tilde{\beta}_{i,j}^{(\kappa)}]_{\kappa=1..m_i}), \text{ where } \tilde{\beta}_{i,j}^{(\kappa)} = u^{(\kappa)} \mathcal{N}(c_{i,j} \ ; \ \eta_i^{(\kappa)}, \phi_i^{(\kappa)}) / \mathbb{P}_{C_i}(c_{i,j})_.$$
3. Sample $k_{i,j} \sim \mathrm{Cat}(k; [\tilde{\beta}_{i,j}^{(\kappa)}]_{\kappa=1..m_i})$ and convert it into an one-hot vector $\beta_{i,j}$.
4. Normalize $c_{i,j}$ as $\alpha_{i,j} = (c_{i,j} - \eta_i^{(k)})/4\phi_i^{(k)}$. Then clip $\alpha_{i,j}$ to $[-1, 1]$, i.e. keep the 4ϕ area of a Gaussian distribution which covers 99.99% of samples. Finally, employ $\alpha_{i,j}$ and $\beta_{i,j}$ to represent $c_{i,j}$.

4.2 Conditional Tabular GAN

Traditionally, a GAN is fed with a vector sampled from a standard multivariate normal distribution (MVN), and by means of the *Generator* and *Discriminator* or *Critic* (Arjovsky et al., 2017; Gulrajani et al., 2017) neural networks one eventually obtains a deterministic transformation that maps the standard MVN into the distribution of the data. This method of training a generator does not account for the imbalance in the categorical columns. If the training data are randomly sampled during training, the rows that fall into the minor category will not be sufficiently represented, thus the generator may not be trained correctly. This problem is reminiscent of the "*class imbalance*" problem in discriminatory modeling - the challenge however is exacerbated since there is not a single column to balance and the real data distribution should be kept intact. If the training data are resampled, the generator learns the resampled distribution which is different from the real data distribution.

Specifically, the goal is to resample efficiently in a way that all the categories from discrete attributes are sampled evenly (but not necessary uniformly) during the training process, and to recover the (not-resampled) real data distribution during test. A way to attain this is to enforce that the generator matches a given category. Let k^* be the value from the i^*th discrete column D_{i^*} that has to be matched by the generated samples $\hat{\mathbf{r}}$, then the generator can be interpreted as the conditional distribution of rows given that particular value at that particular column, i.e. $\hat{\mathbf{r}} \sim \mathbb{P}_{\mathcal{G}}(\text{row}|D_{i*} = k^*)$. For this reason, in this paper we name it *Conditional generator*, and a GAN built upon it is referred to as *Conditional GAN*. Moreover,

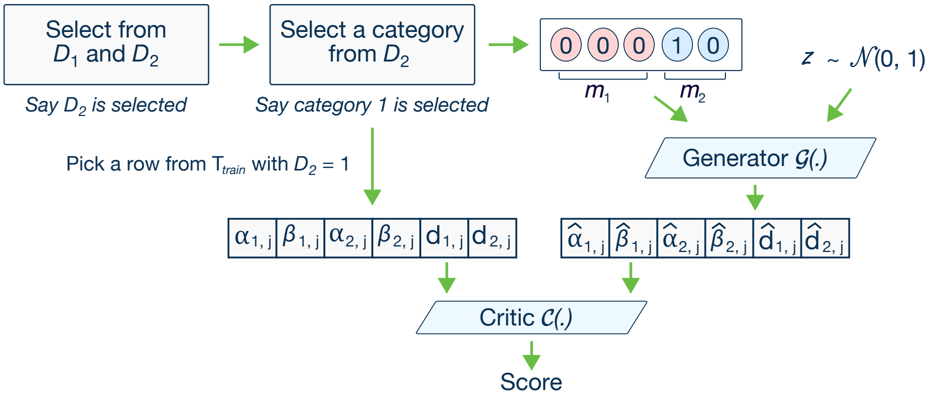

Select from
D_1 and D_2
Say D_2 is selected
Select a category
from D_2
Say category 1 is selected
0 0 0 1 0
m_1
m_2
$z \sim \mathcal{N}(0, 1)$
Generator $\mathcal{G}(.)$
Pick a row from T_{train} with $D_2 = 1$
$\alpha_{1,j}$ $\beta_{1,j}$ $\alpha_{2,j}$ $\beta_{2,j}$ $d_{1,j}$ $d_{2,j}$
$\hat{\alpha}_{1,j}$ $\hat{\beta}_{1,j}$ $\hat{\alpha}_{2,j}$ $\hat{\beta}_{2,j}$ $\hat{d}_{1,j}$ $\hat{d}_{2,j}$
Critic $\mathcal{C}(.)$
Score

Figure 2: TGAN structure.

in this paper we construct our `TGAN` as a Conditional GAN, upon two main modules: the conditional generator $\mathcal{G}$ and the critic $\mathcal{C}$.

Integrating a conditional generator into the architecture of a GAN requires to deal with the following issues: 1) it is necessary to devise a representation for the condition as well as to prepare an input for it, 2) it is necessary for the generated rows to preserve the condition as it is given, and 3) it is necessary for the conditional generator to learn the real data conditional distribution, i.e. $\mathbb{P}_{\mathcal{G}}(\text{row}|D_{i*} = k^*) = \mathbb{P}(\text{row}|D_{i*} = k^*)$, so that

$$\mathbb{P}(\text{row}) = \sum_{k \in D_{i^*}} \mathbb{P}_{\mathcal{G}}(\text{row}|D_{i*} = k^*)\mathbb{P}(D_{i^*} = k).$$

We present a solution that consists of three key elements, namely: the *conditional vector*, the generator loss, and the *training-by-sampling* method.

Conditional vector. We introduce the vector *cond* as the way for indicating the condition $(D_{i^*} = k^*)$. Recall that, after the reversible data transformation, all the discrete columns $D_1, \ldots, D_{N_d}$ end up as one-hot vectors $\mathbf{d}_1, \ldots, \mathbf{d}_{N_d}$ such that the ith one-hot vector is $\mathbf{d}_i = [\mathbf{d}_i^{(k)}]$, for $k = 1, \ldots, |D_i|$. Let $\mathbf{m}_i = [\mathbf{m}_i^{(k)}]$, for $k = 1, \ldots, |D_i|$ be the ith *mask* vector associated to the ith one-hot vector $\mathbf{d}_i$. Hence, the condition can be expressed in terms of these mask vectors as

$$\mathbf{m}_i^{(k)} = \begin{cases} 1 & \text{if} \quad i = i^* \quad \text{and} \quad k = k^*, \\ 0 & \text{otherwise.} \end{cases}$$

Then, define the vector *cond* as $cond = \mathbf{m}_1 \oplus \ldots \oplus \mathbf{m}_{N_d}$. For instance, for two discrete columns, $D_1 = \{1, 2, 3\}$ and $D_2 = \{1, 2\}$,the condition $(D_2 = 1)$ is expressed by the mask vectors $\mathbf{m}_1 = [0, 0, 0]$ and $\mathbf{m}_2 = [1, 0]$; so $cond = [0, 0, 0, 1, 0]$.

Generator loss. During training, the conditional generator is free to produce any set of one-hot discrete vectors $\{\hat{\mathbf{d}}_1, \ldots, \hat{\mathbf{d}}_{N_d}\}$. In particular, given the condition $(D_{i^*} = k^*)$ in the form of *cond* vector, nothing in the feed-forward pass prevents from producing either $\hat{\mathbf{d}}_{i^*}^{(k^*)} = 0$ or $\hat{\mathbf{d}}_{i^*}^{(k)} = 1$ for $k \neq k^*$. The mechanism proposed to enforce the conditional generator to produce $\hat{\mathbf{d}}_{i^*} = \mathbf{m}_{i^*}$ is to penalize its loss by adding the cross-entropy between

$\mathbf{m}_{i^*}$ and $\hat{\mathbf{d}}_{i^*}$, averaged over all the instances of the batch. Thus, as the training advances, the generator learns to make an exact copy of the given $\mathbf{m}_{i^*}$ into $\hat{\mathbf{d}}_{i^*}$.

Training-by-sampling. The output produced by the conditional generator must be assessed by the critic, which estimates the distance between the learned conditional distribution $\mathbb{P}_{\mathcal{G}}(\text{row}|\mathit{cond})$ and the conditional distribution on real data $\mathbb{P}(\text{row}|\mathit{cond})$. The sampling of real training data and the construction of *cond* vector should comply to help critic estimate the distance. There are two possibilities: either we randomly select an instance (row) from the table and then select the condition attribute in it, or we randomly select an attribute (column) and a value from that column and then select a row filtering the table by the value of that column. Clearly, the first one is not appropriate for our goal because we cannot ensure that all the values from discrete attributes are sampled evenly during the training process. On the other hand, if we consider all the discrete columns equally likely and randomly select one, and then consider all the values in its range equally likely, it might be the case that one row from a very low frequency category will be excessively oversampled; so once again is not an appropriate choice. Thus, for our purposes, we propose the following steps:

1. Create N_d zero-filled mask vectors $\mathbf{m}_i = [\mathbf{m}_i^{(k)}]_{k=1\ldots|D_i|}$, for $i = 1, \ldots, N_d$, so the ith mask vector corresponds to the ith column, and each component is associated to the category of that column.
2. Randomly select a discrete column D_i out of all the N_d discrete columns, with equal probability. Let i^* be the index of the column selected. For instance, in Figure 2, the selected column was D_2, so $i^* = 2$.
3. Construct a PMF across the range of values of the column selected in 2, D_{i^*}, such that the probability mass of each value is the logarithm of its frequency in that column.
4. Let k^* be a randomly selected value according to the PMF above. For instance, in Figure 2, the range D_2 has two values and the first one was selected, so $k^* = 1$.
5. Set the k^*th component of the i^*th mask to one, i.e. $\mathbf{m}_{i^*}^{(k^*)} = 1$.
6. Calculate the vector $\mathit{cond} = \mathbf{m}_1 \oplus \cdots \mathbf{m}_{i^*} \oplus \mathbf{m}_{N_d}$. For instance, in Figure 2, we have the masks $\mathbf{m}_1 = [0, 0, 0]$ and $\mathbf{m}_{2*} = [1, 0]$, so $\mathit{cond} = [0, 0, 0, 1, 0]$.

We use the PacGAN framework (Lin et al., 2018), taking 10 samples from training data in each pac. The training algorithm under this framework is completely described in Algorithm 1. It begins by creating as many condition vectors *cond*, and drawing as many samples from standard MVN, as the batch size (lines 1-3). Both are fed-forward into the conditional generator to produce a batch of *fake* rows (line 4). The input to PacGAN is twofold. On one hand, it comes from sampling the training tabular data according to the *cond* vector. On the other hand, it is the output of the conditional generator. Both are preprocessed as detailed in lines 7 and 8 before being fed-forwarded into the critic, to obtain its loss $\mathcal{L}_C$ (line 9). In lines 10-12 we follow (Gulrajani et al., 2017) to compute the gradient penalty for the critic. To update the parameters of the critic we use a gradient descent step, with learning rate $2 \cdot 10^{-4}, \beta_1 = 0.9, \beta_2 = 0.5$ and Adam optimizer (line 13). In order to update the parameters of the conditional generator, it is first necessary to repeat the feed-forward steps both in the conditional generator (lines 1-7) and in the critic (line 15) , which leads to the loss of the conditional generator, since in this step the critic is not updated. Then, we use a gradient descent step similar to the one for the parameters of the critic (line 16).

Finally, the conditional generator $\mathcal{G}(z, cond)$ architecture can be formally described as

$$\begin{cases} h_1 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|cond|+|z|\to 256}(z \oplus cond))) & \\ h_2 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|cond|+|z|+256\to 256}(z \oplus cond \oplus h_1))) & \\ \hat{\alpha}_i = \texttt{tanh}(\texttt{FC}_{|cond|+|z|+512\to 1}(h_2)) & 1 \le i \le N_c \\ \hat{\beta}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{|cond|+|z|+512\to m_i}(h_2)) & 1 \le i \le N_c \\ \hat{\mathbf{d}}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{|cond|+|z|+512\to |D_i|}(h_2)) & 1 \le i \le N_d \end{cases}$$

and, the architecture of the critic (with pac size 10) $\mathcal{C}(\mathbf{r}_1, \ldots, \mathbf{r}_{10}, cond_1, \ldots, cond_{10})$ can be formally described as

$$\begin{cases} h_0 = \mathbf{r}_1 \oplus \ldots \oplus \mathbf{r}_{10} \oplus cond_1 \oplus \ldots \oplus cond_{10} \\ h_1 = \texttt{drop}(\texttt{leaky}_{0.2}(\texttt{FC}_{10|\mathbf{r}|+10|cond|\to 256}(h_0))) \\ h_2 = \texttt{drop}(\texttt{leaky}_{0.2}(\texttt{FC}_{256\to 256}(h_1))) \\ \mathcal{C}(\cdot) = \texttt{FC}_{256\to 1}(h_2) \end{cases}$$

Generate synthetic data for different purposes. During testing, the user has to provide the Conditional `TGAN` both with a random MVN vector z (as to any other GAN) and a *cond* vector properly constructed according to the discrete columns and their range of values. Users can construct *cond* to generate rows with a specific value in a discrete column, for exmaple generate several columns with $D_2 = 1$. In our experiments, i^* is sampled uniformly and $\mathbf{m}_{i^*}$ follows the marginal distribution of D_{i^*} so that the generated data are expected to reveal the real data distribution.

5. Benchmarking synthetic data generation algorithms

There are multiple deep learning methods for modeling tabular data. We notice that all methods and their corresponding papers neither employed the same datasets nor were evaluated under similar metrics. This fact made comparison challenging and did not allow for identifying each method's weaknesses and strengths *vis-a-vis* the intrinsic challenges presented when modeling tabular data. To address this, we developed a comprehensive benchmarking suite.

5.1 Baselines and datasets

Our baselines consist of Bayesian networks (`CLBN` (Chow and Liu, 1968), `PrivBN` (Zhang et al., 2017)), and implementations of current deep learning approaches for synthetic data generation (`MedGAN` (Choi et al., 2017), `VeeGAN` (Srivastava et al., 2017), `TableGAN` (Park et al., 2018)). This library along with its very easy to use APIs are described in the supplementary material. More datasets and methods can be easily added.

To challenge comparisons and motivate further development, we added a VAE baseline as well, called `TVAE`. `TVAE` uses the same preprocessing as `TGAN`. The structure and loss function of VAE have been adapted accordingly so as to model tabular data (details can be found in supplemental materials.)

```
Algorithm 1: Train TGAN on step.
  Input: Training data $\mathbf{T}_{train}$; Conditional generator parameters $\Phi_G$; Critic parameters $\Phi_C$; batch size $m$; pac size $pac$.
  Result: Conditional generator and Critic parameters $\Phi_G$ and $\Phi_C$ updated.
1  Create masks $\{\mathbf{m}_1, \ldots, \mathbf{m}_{i^*}, \ldots, \mathbf{m}_{N_d}\}_j$,   for $1 \le j \le m$
2  Create condition vectors $cond_j$,   for $1 \le j \le m$ from masks   ▷ Create $m$ conditional vectors
3  Sample $\{z_j\} \sim \texttt{MVN}(0, \mathbf{I})$ ,   for $1 \le j \le m$
4  $\hat{\mathbf{r}}_j \leftarrow \texttt{Generator}(z_j, cond_j)$ ,   for $1 \le j \le m$   ▷ Generate fake data
5  Sample $\mathbf{r}_j \sim \texttt{Uniform}(\mathbf{T}_{train}|cond_j)$ ,   for $1 \le j \le m$   ▷ Get real data
6  $cond_k^{(pac)} \leftarrow cond_{k\times pac+1} \oplus \ldots \oplus cond_{k\times pac+pac}$,   for $1 \le k \le m/pac$   ▷ Conditional vector pacs
7  $\hat{\mathbf{r}}_k^{(pac)} \leftarrow \hat{\mathbf{r}}_{k\times pac+1} \oplus \ldots \oplus \hat{\mathbf{r}}_{k\times pac+pac}$ ,   for $1 \le k \le m/pac$   ▷ Fake data pacs
8  $\mathbf{r}_k^{(pac)} \leftarrow \mathbf{r}_{k\times pac+1} \oplus \ldots \oplus \mathbf{r}_{k\times pac+pac}$ ,   for $1 \le k \le m/pac$   ▷ Real data pacs
9  $\mathcal{L}_C \leftarrow \frac{1}{m/pac}\sum_{k=1}^{m/pac} \texttt{Critic}(\hat{\mathbf{r}}_k^{(pac)}, cond_k^{(pac)}) - \frac{1}{m/pac}\sum_{k=1}^{m/pac} \texttt{Critic}(\mathbf{r}_k^{(pac)}, cond_k^{(pac)})$
10 Sample $\rho_1, \ldots, \rho_{m/pac} \sim \texttt{Uniform}(0, 1)$
11 $\tilde{\mathbf{r}}_k^{(pac)} \leftarrow \rho_k \hat{\mathbf{r}}_k^{(pac)} + (1 - \rho_k)\mathbf{r}_k^{(pac)}$ ,   for $1 \le k \le m/pac$
12 $\mathcal{L}_{GP} \leftarrow \frac{1}{m/pac}\sum_{k=1}^{m/pac}(||\nabla_{\tilde{\mathbf{r}}_k^{(pac)}} \texttt{Critic}(\tilde{\mathbf{r}}_k^{(pac)}, cond_k^{(pac)})||_2 - 1)^2$   ▷ Gradient Penalty (Gulrajani et al., 2017)
13 $\Phi_C \leftarrow \Phi_C - 0.0002 \times \texttt{Adam}(\nabla_{\Phi_C}(\mathcal{L}_C + 10\mathcal{L}_{GP}))$
14 Regenerate $\hat{\mathbf{r}}_j$ following lines 1 to 7
15 $\mathcal{L}_G \leftarrow -\frac{1}{m/pac}\sum_{k=1}^{m/pac} \texttt{Critic}(\hat{\mathbf{r}}_k^{(pac)}, cond_k^{(pac)}) + \frac{1}{m}\sum_{j=1}^{m} \texttt{CrossEntropy}(\hat{\mathbf{d}}_{i^*,j}, \mathbf{m}_{i^*})$
16 $\Phi_G \leftarrow \Phi_G - 0.0002 \times \texttt{Adam}(\nabla_{\Phi_G}\mathcal{L}_G)$
```

Simulated data We handcrafted a simulated data oracle $\mathcal{S}$ to represent a known joint distribution, then sample $\mathbf{T}_{train}$ and $\mathbf{T}_{test}$ from $\mathcal{S}$. This oracle is a Gaussian mixture model or a Bayesian network. We followed (Srivastava et al., 2017) to generate `Grid` and `Ring` Gaussian mixture oracles. We add random offset to each mode in `Grid` and call it `GridR`. We pick 4 well known Bayesian networks - `alarm`, `child`, `asia`, `insurance`,[3] - and construct Bayesian network oracles.

Real datasets: We picked 6 commonly used machine learning feature-and-label tables, `adult`, `census`, `covertype`, `intrusion` and `news` from UCI machine learning repository (Dua and Graff, 2017) and `credit` from Kaggle. We also binarized 28 × 28 the MNIST (LeCun and Cortes, 2010) dataset and converted each sample to 784 dimensional vector plus one label column to mimic high dimensional binary data, called `MNIST28`. We resized the images to 12 × 12 and used the same process to generate `MNIST12`.

[3] The structure of Bayesian networks can be found at `http://www.bnlearn.com/bnrepository/`.

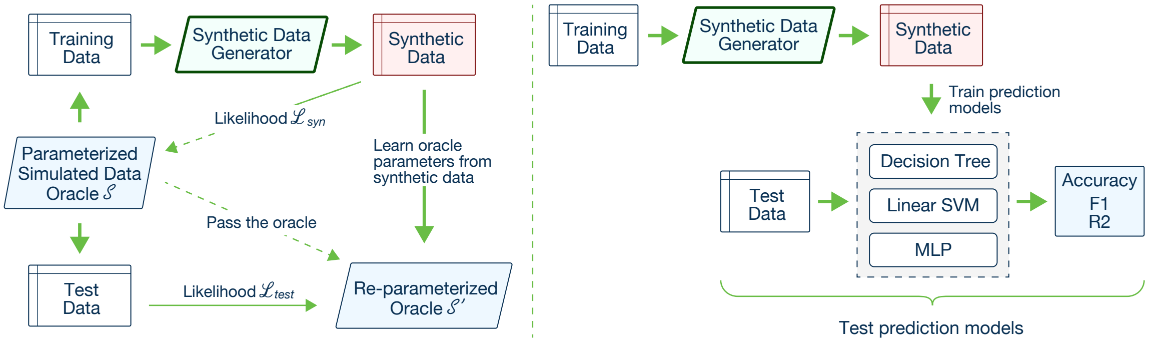


Figure 3: Evaluation framework on simulated data (left) and real data (right).

5.2 Evaluation metrics

Given that evaluation of generative models is not a straightforward process, where different metrics yield substantially diverse results (Theis et al., 2016), our benchmark evaluates multiple metrics on multiple datasets. Simulated data have known probability distribution and are used to evaluate the likelihood fitness, whereas real datasets come from a real machine learning task and can be used to evaluate the machine learning efficacy. Figure 3 illustrates the evaluation framework.

Likelihood fitness metric: On simulated data, take advantage of simulated data oracle $\mathcal{S}$ to compute the `likelihood fitness` metric. We retrain the simulated data generator $\mathcal{S}'$ using $\mathbf{T}_{syn}$. S' has the same structure but different parameters as S. If $\mathcal{S}$ is a Gaussian mixture model, we use the same number of Gaussian components and retrain the mean and covariance of each component. If $\mathcal{S}$ is a Bayesian network, we keep the same graphical structure and learn a new conditional distribution on each edge. We compute the likelihood of $\mathbf{T}_{test}$ on $\mathcal{S}'$. This metric overcomes the issue in $\mathcal{L}_{syn}$. It can detect mode collapse. But this metric introduces the prior knowledge of the structure of $\mathcal{S}'$ which is not necessarily encoded in $\mathbf{T}_{syn}$.

Machine learning efficacy: For a real dataset, we cannot compute the likelihood fitness, instead we evaluate the performance of using synthetic data as training data for machine learning. We train prediction models on $\mathbf{T}_{syn}$ and test prediction models using $\mathbf{T}_{test}$. We evaluate the performance of classification tasks using accuracy and F1, and evaluate the regression task using R^2. For each dataset, we select classifiers or regressors that achieve reasonable performance on each data. (Models and hyperparameters can be found in supplementary material as well as our benchmark framework.) Since we are not trying to pick the best classification or regression model, we take the the average performance of multiple prediction models as metrics for $\mathcal{G}$.

6. Experiments and results

We evaluate `CLBN`, `PrivBN`, `MedGAN`, `VeeGAN`, `TableGAN`, `TGAN`, and `TVAE` using our benchmark framework. We train each model with a batch size of 500. Each model is trained for 300 epochs. Each epoch contains $N/batch\_size$ steps where N is the number of rows in the training set. For `TGAN`, we use hyperparameters described in section 4. Hyperparameters

Table 3: Benchmark results over three sets of experiments, namely Gaussian mixture simulated data, Bayesian network simulated data, and real data. The number in the bracket is the rank of a method (lower better). It is computed as follows: For each set of experiment, (1) rank algorithms over all metrics in each set. (2) Take the average of all ranks of each algorithm. Get one score in range $[1, 7]$ for each algorithm. (3) Rank the score again.

	grid		gridr		ring	
method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$
Identity	-3.06	-3.06	-3.06	-3.07	-1.70	-1.70
CLBN(2)	-3.68	-8.62	-3.76	-11.60	-1.75	**-1.70**
PrivBN(4)	-4.33	-21.67	-3.98	-13.88	-1.82	-1.71
MedGAN(7)	-10.04	-62.93	-9.45	-72.00	-2.32	-45.16
VEEGAN(6)	-9.81	-4.79	-12.51	-4.94	-7.85	-2.92
TableGAN(5)	-8.70	-4.99	-9.64	-4.70	-6.38	-2.66
TVAE(1)	**-2.86**	-11.26	**-3.41**	**-3.20**	**-1.68**	-1.79
TGAN(3)	-5.63	**-3.69**	-8.11	-4.31	-3.43	-2.19

	asia		alarm		child		insurance	
method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$
Identity	-2.23	-2.24	-10.3	-10.3	-12.0	-12.0	-12.8	-12.9
CLBN(3)	-2.44	-2.27	-12.4	-11.2	-12.6	-12.3	-15.2	-13.9
PrivBN(1)	**-2.28**	**-2.24**	-11.9	-10.9	-12.3	**-12.2**	-14.7	**-13.6**
MedGAN(5)	-2.81	-2.59	**-10.9**	-14.2	-14.2	-15.4	-16.4	-16.4
VEEGAN(7)	-8.11	-4.63	-17.7	-14.9	-17.6	-17.8	-18.2	-18.1
TableGAN(6)	-3.64	-2.77	-12.7	-11.5	-15.0	-13.3	-16.0	-14.3
TVAE(2)	-2.31	-2.27	-11.2	**-10.7**	**-12.3**	-12.3	**-14.7**	-14.2
TGAN(4)	-2.56	-2.31	-14.2	-12.6	-13.4	-12.7	-16.5	-14.8

	adult	census	credit	cover.	intru.	mnist12/28		news
method	F1	F1	F1	Macro	Macro	Acc	Acc	R^2
Identity	0.669	0.494	0.720	0.652	0.862	0.886	0.916	0.14
CLBN(3)	0.334	0.310	0.409	0.319	0.384	0.741	0.176	-6.28
PrivBN(4)	0.414	0.121	0.185	0.270	0.384	0.117	0.081	-4.49
MedGAN(6)	0.375	0.000	0.000	0.093	0.299	0.091	0.104	-8.80
VEEGAN(6)	0.235	0.094	0.000	0.082	0.261	0.194	0.136	-6.5e6
TableGAN(5)	0.492	0.358	0.182	0.000	0.000	0.100	0.000	-3.09
TVAE(1)	**0.626**	0.377	0.098	**0.433**	0.511	**0.793**	**0.794**	**-0.20**
TGAN(1)	0.601	**0.391**	**0.672**	0.324	**0.528**	0.394	0.371	-0.43

for TVAE can be found in supplementary materials. We posit that for any dataset, across any metrics except $\mathcal{L}_{syn}$, the best performance is achieved by $\mathbf{T}_{train}$. Thus we present the Identity method which outputs $\mathbf{T}_{train}$.

Experimental results are shown in Table 3. In the continuous data case, CLBN and PrivBN suffer because continuous data are discretized. MedGAN, VeeGAN, and TableGAN all suffer from mode collapse. With mode-specific normalization, our model performs well on 2D continuous datasets.
On dataset generated from Bayesian networks, CLBN and PrivBN have a natural advantage. Our TGAN achieves slightly better performance than MedGAN and TableGAN. Surprisingly, TableGAN works well on discrete datasets, despite considering discrete columns as continuous values. Our reasoning for this is that in our simulated data, most columns have fewer than 4 categories, so conversion does not cause serious problems. On real datasets, TVAE and TGAN outperforms CLBN and PrivBN, whereas other GAN models cannot get as good result as Bayesian networks. With respect to large scale real datasets, learning a high-quality Bayesian network is difficult. There is a significant performance gap between real data and synthetic data generated by a learned Bayesian network.
TVAE outperforms TGAN in several cases, but GANs do have several favorable attributes, and this does not indicate that we should always use VAEs rather than GANs on modeling tables. The GANs generator does not have access to real data during the entire training process; thus, we can make TGAN achieve differential privacy easier than TVAE.

7. Conclusion

In this paper we attempt to find a flexible and robust model to learn the distribution of columns with complicated distributions. We observe that none of the existing deep generative models can outperform Bayesian networks which discretize continuous values and learn greedily. We show several properties that make this task unique and propose our TGAN model. Empirically, we show that our model can learn a better distributions than Bayesian networks. As future work, we would derive a theoretical justification on why GANs can work on a distribution with both discrete and continuous data.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017.

Laura Aviñó, Matteo Ruffini, and Ricard Gavaldà. Generating synthetic but plausible healthcare record datasets. In *KDD workshop on Machine Learning for Medicine and Healthcare*, 2018.

Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. In *ICML workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Zhengping Che, Yu Cheng, Shuangfei Zhai, Zhaonan Sun, and Yan Liu. Boosting deep learning risk prediction with generative adversarial networks for electronic health records. In *International Conference on Data Mining.* IEEE, 2017.

Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Machine Learning for Healthcare Conference*. PMLR, 2017.

C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. Differentially private spatial decompositions. In *International Conference on Data Engineering*. IEEE, 2012.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on International Conference on Machine Learning*, 2015.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2016.

James Jordon, Jinsung Yoon, and Mihaela van der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2013.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database, 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2018.

Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. In *International Conference on Very Large Data Bases*, 2018.

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *International Conference on Data Science and Advanced Analytics*. IEEE, 2016.

Jerome P Reiter. Using cart to generate partially synthetic public use microdata. *Journal of Official Statistics*, 21(3):441, 2005.

Akash Srivastava, Lazar Valkov, 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*, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Yi Sun, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Learning vine copula models for synthetic data generation. In *AAAI Conference on Artificial Intelligence*, 2018.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016.

Alexandre Yahi, Rami Vanguri, Noémie Elhadad, and Nicholas P Tatonetti. Generative adversarial networks for electronic health records: A framework for exploring and evaluating methods for predicting drug-induced laboratory test trajectories. In *NIPS workshop on machine learning for health care*, 2017.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI Conference on Artificial Intelligence*, 2017.

Jun Zhang, Xiaokui Xiao, and Xing Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *International Conference on Management of Data*. ACM, 2016.

Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems*, 42(4):25, 2017.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *international conference on computer vision*, pages 2223–2232. IEEE, 2017.

Table 4: Notations

Notation	Description
	Notations for tabular data generation task.
$C_1, \ldots, C_{N_c}$	N_c continuous columns in tabular data.
$D_1, \ldots, D_{N_d}$	N_d discrete columns in tabular data.
$\mathbf{T}$	Real or simulated tabular data with $N_c + N_d$ columns.
$\mathbf{T}_{train}$, $\mathbf{T}_{test}$	The traning and testing part of $\mathbf{T}$.
$\mathbf{T}_{syn}$	Synthetic data generated by some generative model.
$c_{i,j}$	A float showing the i-th continuous column of j-th row in $\mathbf{T}_{train}$.
$d_{i,j}$	A integer showing the i-th discrete column of j-th row in $\mathbf{T}_{train}$.
	Notations for the benchmark.
$\mathcal{S}, \mathcal{S}'$	Simulated data generator and the retrained generator in the benchmark.
G	A synthetic data generation model to be evaluated.
	Notations for preprocessing.
$\mathbb{P}_{C_i}$	Probabilistic modeling as a Gaussian mixture of continuous column C_i.
m_i	Number of Gaussian components for i-th continuous column.
$u^{(k)}$	Weight of the kth component.
$\eta_i^{(k)}, \phi_i^{(k)}$	The mean and standard deviation of k-th component for i-th continuous column.
$\alpha_{i,j}$	A normalized value for $c_{i,j}$ row.
$\beta_{i,j}$	A one-hot vector denoting the mode $c_{i,j}$ coming from m_i Gaussian distributions.
$\mathbf{d}_{i,j}$	A one-hot representation for $d_{i,j}$.
	Notations for `TGAN`.
$\mathcal{G}(\cdot)$	Generator.
$\mathcal{C}(\cdot)$	Critic.
z	Random noise input.

Table 5: Datasets in our benchmark.

Simulated Data					Real Data					
name	**#train/test**	**#C**	**#B**	**#M**	**name**	**#train/test**	**#C**	**#B**	**#M**	**task**
grid	10k/10k	2	0	0	adult	23k/10k	6	2	7	C
gridr	10k/10k	2	0	0	census	200k/100k	7	3	31	C
ring	10k/10k	2	0	0	covertype	481k/100k	10	44	1	C
asia	10k/10k	0	8	0	credit	264k/20k	29	1	0	C
alarm	10k/10k	0	13	24	intrusion	394k/100k	26	5	10	C
child	10k/10k	0	8	12	mnist12	60k/10k	0	144	1	C
insurance	10k/10k	0	8	19	mnist28	60k/10k	0	784	1	C
					news	31k/8k	45	14	0	R

#C, #B, and #M mean number of continuous columns, binary columns and multi-class discrete columns respectively. C, and R in task mean likelihood, classification and regression respectively.

8. Details about Benchmark

The statistical information of simulated and real data is in Table 5. The raw data of 8 real datasets are avialable online.

- Adult: `http://archive.ics.uci.edu/ml/datasets/adult`
- Census: `https://archive.ics.uci.edu/ml/datasets/census+income`
- Covertype: `https://archive.ics.uci.edu/ml/datasets/covertype`
- Credit: `https://www.kaggle.com/mlg-ulb/creditcardfraud`
- Intrusion: `http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data`
- MNIST: `http://yann.lecun.com/exdb/mnist/index.html`
- News: `https://archive.ics.uci.edu/ml/datasets/online+news+popularity`

For each dataset, I select a few classifiers or regressors which give reasonable performance on such dataset shown in Table 6.

8.1 Data Format

We converted all the datasets into a float array in the interest of consistency. The array has the same number of rows and columns as the original table. It keeps the exact values as the original table for continuous columns. For discrete columns, each category is converted to an integer index. The array stores the index for each category. A separate metafile is created for each dataset, storing the name of the column, the range of a continuous column, and the index to category mapping for a discrete column.

8.2 Current available methods

We provide several baseline methods in our framework. Some of the methods are not designated to generate tabular data. We make small changes to adapt these methods to all the datasets in the benchmark. The result of experiments can be reproduced using default hyper parameters.

CLBN uses the chow-liu algorithm (Chow and Liu, 1968) to create a tree structure Bayesian network. For continuous columns, we evenly discretize them to 15 bins. We use the implementation in pomegranate package (`https://pomegranate.readthedocs.io/en/latest/index.html`).

PrivBN uses a heuristic method to construct a differentially private Bayesian network (Zhang et al., 2017). We wrap the authors' C++ implementation (`https://sourceforge.net/projects/privbayes/`) into our benchmark framework. For continuous columns, we evenly discretize them to 15 bins. We set privacy budget to 10 which is fairly large for the method to model the data accurately instead of adding too much noise.

MedGAN (Choi et al., 2017) is a GAN-based synthetic data generator. The authors released their implementation (`https://github.com/mp2893/medgan`). But the implementation only support continuous data or binary data. It does not support multi category discrete data

Table 6: Classifiers and regressors selected for each real dataset and corresponding performance.

dataset	name	accuracy	f1	macro_f1	micro_f1	r2
adult	Adaboost (estimator=50)	86.07%	68.03%			
	Decision Tree (depth=20)	79.84%	65.77%			
	Logistic Regression	79.53%	66.06%			
	MLP (50)	85.06%	67.57%			
census	Adaboost (estimator=50)	95.22%	50.75%			
	Decision Tree (depth=30)	90.57%	44.97%			
	MLP (100)	94.30%	52.43%			
covtype	Decision Tree (depth=30)	82.25%		73.62%	82.25%	
	MLP (100)	70.06%		56.78%	70.06%	
credit	Adaboost (estimator=50)	99.93%	76.00%			
	Decision Tree (depth=30)	99.89%	66.67%			
	MLP (100)	99.92%	73.31%			
intrusion	Decision Tree (depth=30)	99.91%		85.82%	99.91%	
	MLP (100)	99.93%		86.65%	99.93%	
mnist12	Decision Tree (depth=30)	84.10%		83.88%	84.10%	
	Logistic Regression	87.29%		87.11%	87.29%	
	MLP (100)	94.40%		94.34%	94.40%	
mnist28	Decision Tree (depth=30)	86.08%		85.89%	86.08%	
	Logistic Regression	91.42%		91.29%	91.42%	
	MLP (100)	97.28%		97.26%	97.28%	
news	Linear Regression					0.1390
	MLP (100)					0.1492

or a mix of data types. We modify the autoencoder to support such data. For simplicity, we assume $c_{i,j}$ are min-max normalized to $[0, 1]$, and $\mathbf{d}_{i,j}$ are one-hot representation for categorical columns. The model contains four components:

- An encoder $E^{(AE)}(\cdot)$ that encodes a row to a dense vector.

$$\begin{cases} \mathbf{r}_j = c_{1,j} \oplus \ldots \oplus c_{N_c,j} \oplus \mathbf{d}_{1,j} \oplus \ldots \oplus \mathbf{d}_{N_d,j} \\ x = \texttt{FC}_{|\mathbf{r}_j| \to 128}(\mathbf{r}_j), \\ E^{(AE)}(\mathbf{r}_j) = x_j. \end{cases}$$

- An decoder $D^{(AE)}(\cdot)$ that decodes a row from a dense vector.

$$\begin{cases} \hat{c}_{i,j} = \texttt{sigmoid}(\texttt{FC}_{128\to 1}(x_j)) & 1 \leq i \leq N_c, \\ \hat{\mathbf{d}}_{i,j} = \texttt{softmax}(\texttt{FC}_{128\to |D_i|}(x_j)) & 1 \leq i \leq N_d, \\ D^{(AE)}(x_j) = \{\hat{c}_{1,j}, \ldots, \hat{c}_{N_c,j}, \hat{\mathbf{d}}_{1,j}, \ldots, \hat{\mathbf{d}}_{N_c,j}\}. \end{cases}$$

- A generator $G(\cdot)$ that project a 128-dimensional Gaussian noise to a row.

$$\begin{cases} z_j \sim \mathcal{N}(0, \mathbf{I}) \\ h_1 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{128\to 128}(z_j))) + z_j \\ G(z_j) = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{128\to 128}(h_1))) \end{cases}$$

- A discriminator $D(\cdot)$ that takes a row and the average over a minibatch of size m as features and predicts whether a row of data is real or fake.

$$\begin{cases} \mathbf{r}_j = c_{1,j} \oplus \ldots \oplus c_{N_c,j} \oplus \mathbf{d}_{1,j} \oplus \ldots \oplus \mathbf{d}_{N_d,j} \\ \bar{c}_i = \frac{1}{m}\sum_{j=1}^{m} c_{i,j} & 1 \leq i \leq N_d \\ \bar{\mathbf{d}}_i = \frac{1}{m}\sum_{j=1}^{m} \mathbf{d}_{i,j} & 1 \leq i \leq N_d \\ \bar{\mathbf{r}} = \bar{c}_{1,j} \oplus \ldots \oplus \bar{c}_{N_c,j} \oplus \bar{\mathbf{d}}_{1,j} \oplus \ldots \oplus \bar{\mathbf{d}}_{N_d,j} \\ h_1 = \texttt{ReLU}(\texttt{FC}_{2|\mathbf{r}_j|\to 256}(\mathbf{r}_j \oplus \bar{\mathbf{r}})) \\ h_2 = \texttt{ReLU}(\texttt{FC}_{256\to 128}(h_1)) \\ D(\mathbf{r}_j) = \texttt{sigmoid}(\texttt{FC}_{128\to 1}(h_2)) \end{cases}$$

The encoder and decoder is pretrained for 200 epochs with batch size 500, by minimizing the autoencoder loss

$$\mathcal{L}_{AE} = \sum_{i=1}^{N_c}(c_{i,j} - \hat{c}_{i,j})^2 + \sum_{i=1}^{N_d} \mathbb{CE}(\mathbf{d}_{i,j}, \hat{\mathbf{d}}_{i,j}).$$

After pretrain, the autoencoder is fixed in the rest of the training period.

The generator and discriminator is trained by minimizing

$$\begin{aligned} \mathcal{L}_D &= -\mathbb{E}_{\mathbf{r}_j \sim \mathbf{T}_{train}}[\log D(\mathbf{r}_j)] - \mathbb{E}_{z_j \sim \mathcal{N}(0,\mathbf{I})}[\log(1 - D(D^{(AE)}(G(z_j))))] \\ \mathcal{L}_G &= -\mathbb{E}_{z_j \sim \mathcal{N}(0,I)}[\log D(D^{(AE)}(G(z_j)))] \end{aligned}$$

Adam optimizer is used. The learning rate is 1e-3 and the l2 weight decay is 1e-3.
VeeGAN (Srivastava et al., 2017) add a reconstructor to GAN to detect mode collapse. It is shown to be useful on `grid` dataset. Thus we adapt this method to other datasets. The authors released their implementation `https://github.com/akashgit/VEEGAN`. For simplicity, we assume $c_{i,j}$ are min-max normalized to $[-1, 1]$.[4] The model contains 3 components:

[4] We observe that normalizing continuous column to $[-1, 1]$ and using `tanh` give better performance than $[0, 1]$ and `sigmoid` in `veegan`.

- The generator $G(\cdot)$ that takes a standard Gaussian noise vector and project it to a row of data:

$$\begin{cases} z_j \sim \mathcal{N}(0, \mathbf{I}) \\ h_1 = \texttt{ReLU}(\texttt{FC}_{32\to128}(z_j)) \\ h_2 = \texttt{ReLU}(\texttt{FC}_{128\to128}(h_1)) \\ \hat{c}_{i,j} = \texttt{tanh}(\texttt{FC}_{128\to1}(h_2)) & 1 \le i \le N_c \\ \hat{\mathbf{d}}_{i,j} = \texttt{softmax}(\texttt{FC}_{128\to|D_i|}(h_2)) & 1 \le i \le N_d \\ G(z_j) = \{\hat{c}_{1,j}, \dots, \hat{c}_{N_c,j}, \hat{\mathbf{d}}_{1,j}, \dots, \hat{\mathbf{d}}_{N_c,j}\} \end{cases}$$

- A discriminator $D(\cdot)$ takes the data and the hidden vector, and try to decide whether it's real or fake. The discriminator $D(\cdot)$ works as follows:

$$\begin{cases} h_1 = \texttt{Drop}_{0.5}(\texttt{ReLU}(\texttt{FC}_{|\mathbf{r}_j|+32\to128}(\mathbf{r}_j \oplus z_j))) \\ D(\mathbf{r}_j, z_j) = \texttt{sigmoid}(\texttt{FC}_{128\to1}(h_1)) \end{cases}$$

- A reconstructor $R(\cdot)$ which reconstructs the hidden vector from data.

$$\begin{cases} h_1 = \texttt{ReLU}(\texttt{FC}_{|\mathbf{r}_j|+128\to128}(\mathbf{r}_j)) \\ h_2 = \texttt{ReLU}(\texttt{FC}_{128\to128}((h_1)) \\ R(\mathbf{r}_j) = \texttt{FC}_{128\to32}(h_2) \end{cases}$$

The discriminator, generator and reconstructor are optimized using

$$\begin{aligned} &\mathcal{L}_D = -\mathbb{E}_{\mathbf{r}_j\sim\mathbf{T}_{train}}[\log D(\mathbf{r}_j, R(\mathbf{r}_j))] - \mathbb{E}_{z_j\sim\mathcal{N}(0,\mathbf{I})}[\log(1 - D(G(z_j), z_j))] \\ &\mathcal{L}_G = -\mathbb{E}_{z_j\sim\mathcal{N}(0,\mathbf{I})}[\log D(G(z_j), z_j) + ||z_j - R(G(z_j))||^2] \\ &\mathcal{L}_R = \mathcal{L}_G \end{aligned}$$

TableGAN (Park et al., 2018) is a data synthesizer using convolutional neural networks. It considers all columns as continuous values. Discrete columns are considered as integers in $\{1, \dots, |D_i|\}$. All columns are min-max normalized to $[-1, 1]$. Here we use x_j to denote the $N_c + N_d$ dimensional vector. x_j is then padded and wrapped to a 16×16 matrix $x_j^{(M)}$. [5] To describe the model, we define two notations

- $\texttt{mask}(x^{(M)})$: replace the entry representing the label column in the tabular data to 0.
- $\texttt{label}(x^{(M)})$: extract the value of label column from the matrix.
- $\mathbb{STD}[\cdot]$: compute the expected standard diviation.

The model contains three components

[5] To adapt larger datasets in our benchmark, the matrix size is automatically selected in $\{4, 8, 16, 24, 32\}$. The structure will be described using 16×16 matrix.

- a generator that uses deconvolution of project a 100 dimensional standard Gaussian noise to a 16 × 16 matrix

$$\begin{cases} z_j \sim \mathcal{N}(0, I) \\ h_1 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{100\to1024}(z_j)\texttt{.reshape}(2, 2, 256))) \\ h_2 = \texttt{ReLU}(\texttt{BN}(\texttt{deconv}_{2\times2\times256\to4\times4\times128}(h_1))) \\ h_3 = \texttt{ReLU}(\texttt{BN}(\texttt{deconv}_{4\times4\times128\to8\times8\times64}(h_2))) \\ G(z_j) = \texttt{tanh}(\texttt{deconv}_{8\times8\times64\to16\times16\times1}(h_3)) \end{cases}$$

- a discriminator is

$$\begin{cases} h_1 = \texttt{leaky}_{0.2}(\texttt{BN}(\texttt{conv}_{16\times16\times1\to8\times8\times64}(x_j^{(M)}))) \\ h_2 = \texttt{leaky}_{0.2}(\texttt{BN}(\texttt{conv}_{8\times8\times64\to4\times4\times128}(h_1)) \\ h_3 = \texttt{leaky}_{0.2}(\texttt{BN}(\texttt{conv}_{4\times4\times128\to2\times2\times256}(h_2))) \\ D(x_j^(M)) = \texttt{sigmoid}(\texttt{FC}_{1024\to1}(h_3)\texttt{.reshape}(1024)) \end{cases}$$

- A classifier that predict the label column from all other columns

$$\begin{cases} h_1 = \texttt{leaky}_{0.2}(\texttt{BN}(\texttt{conv}_{16\times16\times1\to8\times8\times64}(\texttt{mask}(x_j^{(M)})))) \\ h_2 = \texttt{leaky}_{0.2}(\texttt{BN}(\texttt{conv}_{8\times8\times64\to4\times4\times128}(h_1)) \\ h_3 = \texttt{leaky}_{0.2}(\texttt{BN}(\texttt{conv}_{4\times4\times128\to2\times2\times256}(h_2))) \\ D(x_j^(M)) = \texttt{sigmoid}(\texttt{FC}_{1024\to1}(h_3)\texttt{.reshape}(1024)) \end{cases}$$

The discriminator is trained by

$$\mathcal{L}_D = -\mathbb{E}_{x_j^{(M)}\sim\mathbf{T}_{train}}[\log D(x_j^{(M)})] - \mathbb{E}_{z_j\sim\mathcal{N}(0,I)}[\log(1 - D(G(z_j)))].$$

The generator is trained by minimizing the sum of the following loss functions[6]

$$\begin{aligned} \mathcal{L}_G &= -\mathbb{E}_{z_j\sim\mathcal{N}(0,\mathbf{I})}[\log(D(G(z_j)))] \\ \mathcal{L}_{\text{mean}} &= \big|\big|\mathbb{E}_{x_j^{(M)}\sim\mathbf{T}_{train}}[x_j^{(M)}] - \mathbb{E}_{z_j\sim\mathcal{N}(0,I)}[G(z_j)]\big|\big|_1 \\ \mathcal{L}_{\text{std}} &= \big|\big|\mathbb{STD}_{x_j^{(M)}\sim\mathbf{T}_{train}}[x_j^{(M)})] - \mathbb{STD}_{z_j\sim\mathcal{N}(0,I)}[G(z_j)]\big|\big|_1 \\ \mathcal{L}_{\text{clf}}^{(G)} &= \mathbb{E}_{z_j\sim\mathcal{N}(0,\mathbf{I})}[\mathbb{CE}(C(\texttt{mask}(G(z_j))), \texttt{label}(G(z_j))))] \end{aligned}$$

The classifier is trained by

$$\mathcal{L}_{\text{clf}} = \mathbb{E}_{x_j^{(M)}\sim\mathbf{T}_{train}}[\mathbb{CE}(C(\texttt{mask}(x_j^{(M)})), \texttt{label}(x_j^{(M)}))]$$

If the dataset is not a binary classification task, the classifier is disabled and the $\mathcal{L}_{clf}^{(G)}$ is set to 0.

[6] In Park et al. (2018), L2 norm is used for $\mathcal{L}_{\text{mean}}$ and $\mathcal{L}_{\text{std}}$, while L1 norm is used in their implementation.

8.3 TVAE Model

The VAE simultaneously trains a generative model $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$ and an inference model $q_\phi(\mathbf{z}|\mathbf{x})$ by minimizing the evidence lower-bound (ELBO) loss (Kingma and Welling, 2013)

$$\log p_\theta(\mathbf{x}_j) \geq \mathcal{L}(\theta, \phi; \mathbf{x}_j) = \mathbb{E}_{q_\phi(z_j|\mathbf{x}_j)}\big[\log p_\theta(\mathbf{x}_j|z_j)\big] - \mathbb{KL}[q_\phi(z_j|\mathbf{x}_j)||p(z_j)]. \tag{1}$$

where

$$\log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) = \sum_{j=1}^{n} \log p_\theta(\mathbf{x}_j)$$

Usually $p(z_j)$ is multivariate Gaussian distribution $\mathcal{N}(0, \mathbf{I})$. Moreover, $p_\theta(\mathbf{x_j}|\mathbf{z_j})$ and $q_\phi(\mathbf{z_j}|\mathbf{x_j})$ are parameterized using neural networks and optimized using gradient descent.

When using VAE to model rows $\mathbf{r}_j$ in tabular data $\mathbf{T}$, each row is preprocessed as

$$\mathbf{r}_j = \mathtt{cat}(\alpha_{1,j}, \beta_{1,j}, \dots, \alpha_{N_c,j}, \beta_{N_c,j}, \mathbf{d}_{1,j}, \dots, \mathbf{d}_{N_d,j}),$$

and that affects the design of the network $p_\theta(\mathbf{r}_j|z_j)$ that needs to be done differently so that $p_\theta(\mathbf{r}_j|z_j)$ can be modeled accurately and trained effectively. In our design, the neural network outputs a joint distribution of $2N_c + N_d$ variables, corresponding to $2N_c + N_d$ variables $\mathbf{r}_j$. We assume $\alpha_{i,j}$ follows a Gaussian distribution with different means and variance. All $\beta_{i,j}$ and $\mathbf{d}_{i,j}$ follow a categorical PMF. Here is our design.

$$\begin{cases} h_1 = \mathtt{ReLU}(\mathtt{FC}_{128\to128}(z_j)) & \\ h_2 = \mathtt{ReLU}(\mathtt{FC}_{128\to128}(h_1)) & \\ \bar{\alpha}_{i,j} = \mathtt{tanh}(\mathtt{FC}_{128\to1}(h_2)) & 1 \leq i \leq N_c \\ \hat{\alpha}_{i,j} \sim \mathcal{N}(\bar{\alpha}_{i,j}, \delta_i) & 1 \leq i \leq N_c \\ \hat{\beta}_{i,j} \sim \text{Cat}(\mathtt{softmax}(\mathtt{FC}_{128\to m_i}(h_2))) & 1 \leq i \leq N_c \\ \hat{\mathbf{d}}_{i,j} \sim \text{Cat}(\mathtt{softmax}(\mathtt{FC}_{128\to|D_i|}(h_2))) & 1 \leq i \leq N_d \\ p_\theta(\mathbf{r}_j|z_j) = \prod_{i=1}^{N_c} \mathbb{P}(\hat{\alpha}_{i,j} = \alpha_{i,j}) \prod_{i=1}^{N_c} \mathbb{P}(\hat{\beta}_{i,j} = \beta_{i,j}) \prod_{i=1}^{N_d} \mathbb{P}(\hat{\alpha}_{i,j} = \alpha_{i,j}) & \end{cases}$$

Here $\hat{\alpha}_{i,j}, \hat{\beta}_{i,j}, \hat{\mathbf{d}}_{i,j}$ are random variables. And $p_\theta(\mathbf{r}_j|z_j)$ is the joint distribution of these variables. Not sure if there are better notations. So $\log p_\theta(\mathbf{r}_j|z_j)$ is

$$\sum_{i=1}^{N_c} \log \frac{1}{\sqrt{2\pi\delta_i}} \exp \frac{(\alpha_{i,j} - \bar{\alpha}_{i,j})}{2\delta_i^2} + \sum_{i=1}^{N_c} \mathbb{CE}(\hat{\beta}_{i,j}, \beta_{i,j}) + \sum_{i=1}^{N_d} \mathbb{CE}(\hat{\mathbf{d}}_{i,j}, \mathbf{d}_{i,j}) + \text{constant}. \tag{2}$$

In $p_\theta(\mathbf{r}_j|z_j)$, weight matrices and δ_i are parameters in the network. These parameters are trained using gradient descent.

The modeling for $q_\phi(z_j|\mathbf{r}_j)$ is similar to conventional VAE.

$$\begin{cases} \mathbf{r}_j = \mathtt{cat}(\alpha_{1,j}, \beta_{1,j}, \dots, \alpha_{N_c,j}, \beta_{N_c,j}, \mathbf{d}_{1,j}, \dots, \mathbf{d}_{N_d,j}) \\ h_1 = \mathtt{ReLU}(\mathtt{FC}_{|\mathbf{r}_j|\to128}(\mathbf{r}_j)) \\ h_2 = \mathtt{ReLU}(\mathtt{FC}_{128\to128}(h_1)) \\ \mu = \mathtt{FC}_{128\to128}(h_2) \\ \sigma = \exp(\frac{1}{2}\mathtt{FC}_{128\to128}(h_2)) \\ q_\phi(z_j|\mathbf{r}_j) \sim \mathcal{N}(\mu, \sigma\mathbf{I}) \end{cases}$$

TVAE is trained using Adam with learning rate 1e-3.