Synthesizing Tabular Data using Generative Adversarial Networks

Lei Xu
LIDS, MIT
Cambridge, MA
leix@mit.edu

Kalyan Veeramachaneni
LIDS, MIT
Cambridge, MA
kalyanv@mit.edu

Abstract

Generative adversarial networks (GANs) implicitly learn the probability distribution of a dataset and can draw samples from the distribution. This paper presents, Tabular GAN (TGAN), a generative adversarial network which can generate tabular data like medical or educational records. Using the power of deep neural networks, TGAN generates high-quality and fully synthetic tables while simultaneously generating discrete and continuous variables. When we evaluate our model on three datasets, we find that TGAN outperforms conventional statistical generative models in both capturing the correlation between columns and scaling up for large datasets.

1 Introduction

Today, organizations are increasingly using machine learning on relational tabular data to intelligently augment processes and workflows usually carried out by humans. According to a recent survey performed by the data science platform KAGGLE, tabular data is the most commonly encountered data type in business, and the second most common format in academia [21]. At the same time, researchers are touting synthetic data for its ability to alleviate a number of common data science concerns, including resolving critical bottlenecks [8, 34], clearing bureaucratic hurdles encountered in data access, and providing a "safe data space" for exploration [9, 33]. Synthetic datasets can be generated to fit specific needs, like testing new tools or creating educational tutorials using them or can eliminate unwanted contingencies when sharing data: for example, a company might give its employees synthetic data to prevent them from having access to data that may pertain to their friends or to celebrities, or it might provide synthetic data to external consultants to eliminate risk in the event of an accidental breach.

During the past decade, synthetic data generation has been accomplished by modeling a joint multivariate probability distribution for a given dataset $\mathbb{P}(\mathbf{D})$ and then sampling from that distribution. Complicated datasets have required more complex distributions: for example, a sequence of events may have been modeled using hidden Markov models, or a set of non-linearly correlated variables could be modeled using *copulas*. Nevertheless, these generative models are restricted by the type of distribution functions available to users, severely limiting the representations that can be used to create generative models and subsequently limiting the fidelity of the synthetic data.

At the same time, some researchers in the statistical sciences community have begun been using randomization-based methods to generate synthetic data [40, 37, 39, 25]. Most of these efforts aimed at enabling data disclosure, simultaneously imputing and disclosing data, and preserving the privacy of the people represented by the data (generally, respondents of a survey). The creation of generative models using neural models like variational auto-encoders [24], and, subsequently, generative adversarial networks (GAN) [16] and their numerous extensions, is appealing in terms of both the performance and flexibility offered in representing data and the promise of generating and manipulating images and natural languages.

Preprint. Work in progress.

In this paper, we develop a synthetic data generator based on generative adversarial networks for tabular data. We focus on generating tabular data with mixed variable types (multinomial/discrete and continuous) and propose TGAN. To achieve this, we use LSTM with attention in order to generate data column by column. To asses, we first statistically evaluate the synthetic data generated by TGAN. We also demonstrate that machine learning models trained on data generated using TGAN can achieve significantly higher performance than models trained on data generated from other competitive data synthesizers that rely on multivariate probabilistic graphical models.

The paper is organized as follows: section 2 presents a brief synopsis of related work in GANs and previous synthetic data synthesizers; section 3 introduces our TGAN model; section 4 and section 5 explain experiment settings and results; and section 6 gives our conclusion.

2 Related Work

Generative Adversarial Networks Since GANs were first proposed, many efforts have been made to speed up and stabilize the training process [42, 1, 18, 4], with application-based studies mainly focusing on generating images. GANs can generate high-quality images [36, 10, 22], and some models can even generate images conditioned on images or texts [28, 38, 20, 51, 23, 48]. However, generating discrete variables is a challenge for GANs: the authors in [26, 7] attempt a differential model by designing special functions or modifying the loss function; other researchers [50, 14, 46] use a reinforcement learning framework to train the non-differentiable model, making natural language generation possible. Other GAN applications include information retrieval [45], dialogue systems [27], and speech processing [32].

Synthetic Data Generation Synthetic data is useful in data science, as shown in Howe et al. [19]'s thorough analysis of such data's use cases and social benefits. There are several statistical ways to generate this kind of synthetic data, including [1] classification and [2] regression trees [41, 29] and [3] Bayesian networks [12, 30, 43]. Ping et al. [35] introduce a web-based Data Synthesizer using a Bayesian network to model the correlation between features. Patki et al. [34] propose a framework to recursively generate a relational database and use *copulas*. Besides statistical models generating fully synthetic data, neural models are used to impute missing values in datasets; for example, Gondara and Wang [15] uses deep de-noising autoencoders, and Yoon et al. [49] use GAN. Recently, several GAN models emerged to handle tabular data, especially to generate medical records. `RGAN` and `RCGAN` [13] can generate real-valued time-series data. `medGAN` [9], `corrGAN` [33] and several improved models [2, 6, 47, 3, 44] can generate discrete medical records but do not tackle the complexity in generating multimodal continuous variables. `ehrGAN` [8] generates augmented medical records but doesn't explicitly generate synthetic data.

Finally, perhaps the work that is closest to our work is `tableGAN` [31] - that is, it tries to solve the problem of generating synthetic data for a tabular dataset[1]. However, there are a few fundamental differences. It uses convolutional neural networks while we use recurrent networks. Also, `tableGAN` explicitly optimizes the prediction accuracy on synthetic data by minimizing cross entropy loss while our model cares more about marginal distribution. We explicitly learn the marginal distribution of each column by minimizing KL divergence.

3 GANs for tabular data

Developing a general-purpose GAN that would reliably work for a tabular dataset is nontrivial. Complexities arise due to the various types of data that can be present in the table, including numerical, categorical, time, text, and cross-table references. This is in addition to the variety of shapes the distributions of these variables can take, including multimodal, long tail, and several others. We begin by formalizing the synthetic table generation task and describing the mechanisms that evaluate how well synthetic data actually achieve the goals described in the previous section.

Synthetic table generation task: A table $\mathbf{T}$ contains n_c continuous random variables - $\{C_1, \ldots, C_{n_c}\}$, and n_d discrete (multinomial) random variables $\{D_1, \ldots, D_{n_d}\}$. These variables follow an unknown joint distribution $\mathbb{P}(C_{1:n_c}, D_{1:n_d})$. Each row is one sample from the joint distribution

[1] `tableGAN` was released in June 2018 and we were informed about its existence during a review process for this paper - submitted in May 2018.

represented using a lowercase $\{c_{1,j}, \ldots, c_{n_c,j}, d_{1,j}, \ldots, d_{n_d,j}\}$. Each row is sampled independently; that is, we do not consider sequential data. The goal is to learn a generative model $\mathtt{M}(C_{1:n_c}, D_{1:n_d})$ such that samples generated from this model $\mathtt{M}$ create a synthetic table $\mathbf{T}_{synth}$ that can satisfy the following requirements. (1) A machine learning model learned using $\mathbf{T}_{synth}$ can achieve a similar accuracy on a real test table $\mathbf{T}_{test}$ (usually set aside at the beginning), as would a model learned using the data from table $\mathbf{T}$. (2) Mutual information: The mutual information between an arbitrary pair of variables i, j in $\mathbf{T}$ and $\mathbf{T}_{\mathbf{synth}}$ is similar.

3.1 Reversible Data Transformation

To enable neural networks to learn the model effectively we apply a series of reversible transformations to the variables in the table. Neural networks can effectively generate values with a distribution centered over $(-1, 1)$ using $\tanh$, as well as a low-cardinality multinomial distribution using softmax. Thus, we convert a numerical variable into a scalar in the range $(-1, 1)$ and a multinomial distribution, and convert a discrete variable into a multinomial distribution.

Mode-specific normalization for numerical variables: Numerical variables in tabular datasets sometimes follow a multimodal distribution. We use a Gaussian kernel density estimation to estimate the number of modes of a continuous variable. In the three datasets we use this paper, we found that $4/7$ variables in the Census dataset, $22/27$ continuous variables in the KDD99 dataset, and $1/10$ variables in the Covertype dataset have multiple modes. Simply normalizing numerical feature to $[-1, 1]$ and using $\tanh$ activation to generate these features does not work well. For example, if there is a mode close to -1 or 1, the gradient will saturate when back-propagating through $\tanh$.

To effectively sample values from a multimodal distribution, we cluster values of a numerical variable using a Gaussian Mixture model (GMM).

- We train a GMM with m components for each numerical variable C_i. GMM models a distribution with a weighted sum of m Gaussian distributions. The means and standard deviations of the m Gaussian distributions are $\eta_i^{(1)}, \ldots, \eta_i^{(m)}$ and $\sigma_i^{(1)}, \ldots, \sigma_i^{(m)}$.
- We compute the probability of $c_{i,j}$ coming from each of the m Gaussian distributions as a vector $u_{i,j}^{(1)}, \ldots u_{i,j}^{(m)}$. $u_{i,j}$ is a normalized probability distribution over m Gaussian distributions.
- We normalize $c_{i,j}$ as $v_{i,j} = (c_{i,j} - \eta_i^{(k)})/2\sigma_i^{(k)}$, where $k = \arg\max_k u_{i,j}^{(k)}$. We then clip $v_{i,j}$ to $[-0.99, 0.99]$.

Then we use u_i and v_i to represent c_i. For simplicity, we cluster all the numerical features, i.e. both uni-modal and multi-modal features are clustered to $m = 5$ Gaussian distributions. The simplification is fair because GMM automatically weighs m components. For example, if a variable has only one mode and fits some Gaussian distribution, then GMM will assign a very low probability to $m - 1$ components and only 1 remaining component actually works, which is equivalent to not clustering this feature.

Smoothing for categorical variables: In generating categorical variables the model faces a similar challenge it faces in natural language generation, which is how to make the model differentiable. In natural language generation, people use reinforcement learning [50] or Gumbel softmax [26] to deal with this issue. We are facing a similar challenge but the number of categories is much smaller than the size of vocabulary in the natural language. So we can generate the probability distribution directly using softmax. But we find it necessary to convert categorical variables to one-hot-encoding representation and add noise to binary variables.

- A sample $d_{i,j}$ of a discrete variable D_i is first represented as a $|D_i|$-dimensional one-hot vector $\mathbf{d}_{i,j}$.
- We then add noise to each dimension as $\mathbf{d}_{i,j}^{(k)} \leftarrow \mathbf{d}_{i,j}^{(k)} + \mathtt{Uniform}(0, \gamma)$. We set $\gamma = 0.2$.
- We then renormalize the representation as $\mathbf{d}_{i,j} \leftarrow \mathbf{d}_{i,j} / \sum_{k=1}^{|D_i|} \mathbf{d}_{i,j}^{(k)}$.

After prepocessing, we convert $\mathbf{T}$ with $n_c + n_d$ columns to $v_{1:n_c,j}, u_{1:n_c,j}, \mathbf{d}_{1:n_d,j}$. The sum of the dimensions of these vectors is $n_c(m+1) + \sum_{i=1}^{n_d} |D_i|$. This vector is the output of the generator

Reversible Data Transformation.

notation 정리.

j번째 Sample에 대한 순은의 notation: $\{c_{1,j}, \dots, c_{n_c,j}, d_{1,j}, \dots, d_{n_d,j}\}$ c_{ij} ; j번째 data의 i번째 feature value.

Mode-specific normalization for numerical variables.

Numerical variables in tabular datasets sometimes follow a multimodal distribution.

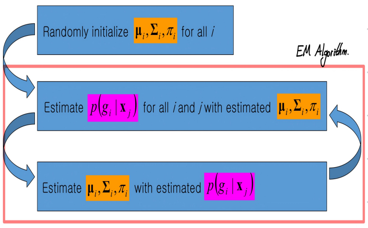
So, simply normalizing numerical feature to [-1, 1] and using tanh activation to generate these features does not work well.

ex) if there is a mode close to -1 or 1, the gradient will saturate.

To effectively sample values from a multimodal distribution, we cluster values of a numerical variable using a Gaussian Mixture Model.

Gaussian Mixture Model. ; Data와 gaussian의 수가 주어진 상황에서 적절한 gaussian들을 찾는 문제이다.

Probability that we observe x ; $P(x) = \sum_{i=1}^{m} P(x|g_i)P(g_i)$ $P(g_i)$; 해당 gaussian(g_i)가 선택될 확률.

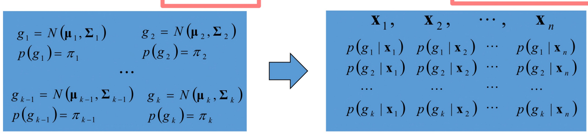


EM Algorithm.

현재 문제 조건: We have data which are generated by m gaussians
We don't know parameters (μ, Σ) and chosen probability of gaussian.
We don't know which Gaussian generates each data.

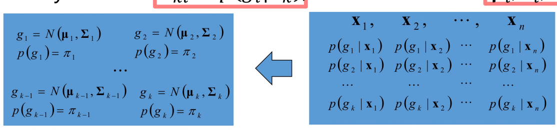
step1. We have data which are generated by m gaussians
We know parameters (μ, Σ) and chosen probability of gaussian. 즉, gaussian의 parameters를 모르지만, 알고있다고 가정한다. 처음에는 엉터리 결과가 나오지만 결국 최적화 될 것이라는 idea.
We don't know which Gaussian generates each data.

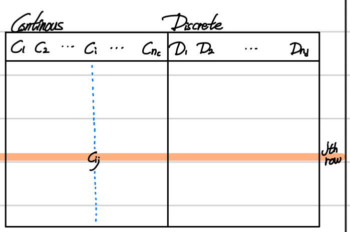
1. Once you know μ_i, Σ_i, π_i, we can estimate $z_{ki} = p(g_i|\mathbf{x}_k)$



step2. We have data which are generated by m gaussians
We don't know parameters (μ, Σ) and chosen probability of gaussian.
We know which Gaussian generates each data.

2. Once you know $z_{ki} = p(g_i|\mathbf{x}_k)$, we can estimate μ_i, Σ_i, π_i



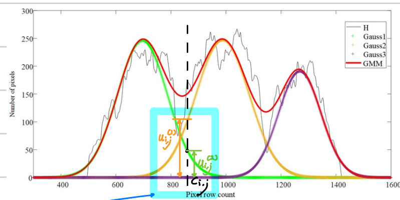


Mode-specific normalization step1.

We train a GMM with m components for each numerical variable C_i.

→ 즉, 우리는 C_i feature의 distribution을 m개의 gaussian들로 표현할 수 있도록 학습하였다.

즉, the means and standard deviations of the m Gaussian distributions $(\eta_i^{(1)}, \eta_i^{(2)}, \cdots \eta_i^{(m)})$, $(\sigma_i^{(1)}, \sigma_i^{(2)}, \cdots, \sigma_i^{(m)})$를 학습함.



Mode-specific normalization step2.

We compute the probability of c_{ij} coming from each of the m Gaussian distributions as a vector $u_{i,j}^{(1)}, u_{i,j}^{(2)}, \cdots u_{i,j}^{(m)}$

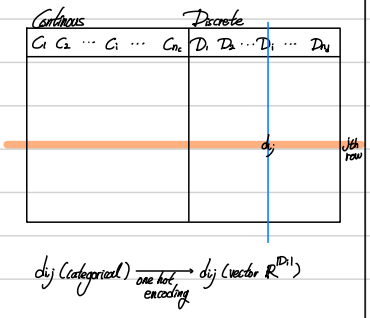
→ 즉, 각 datapoint에서 m개의 Gaussian distribution에 대한 likelihood를 계산하여 vectorization 한다.

Mode-specific normalization step3.

We normalize $c_{i,j}$ as $v_{i,j} = (c_{i,j} - \eta_i^{(k)}) / 2\sigma_i^{(k)}$, where $k = \arg\max_k u_{i,j}^{(k)}$. 즉, 해당 vector에서 가장 likelihood가 큰 gaussian의 mean, std를 이용하여 Normalize하고, [-0.99, 0.99] 값으로 저장한다.

Then we use u_i and v_i to represent c_i.

Reversible Data Transformation.



Smoothing for categorical variables.

We can generate the probability distribution directly using softmax.

But we find it necessary to convert categorical variables to one-hot encoding representation and add noise to binary variables.

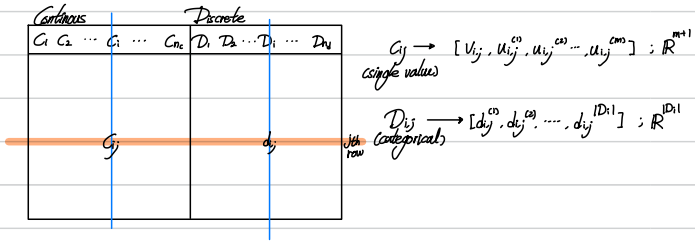
내가 정리할 때 one-hot encoding 결과에 noise를 섞어준 이유는 0, 1 숫자에 numeric 값만 출력하는 것은 방지하기 위함으로 생각한다.

step1. A sample d_{ij} of a discrete variable D_i is first represented as a $|D_i|$-dimensional one-hot vector d_{ij}.

step2. We then add noise to each dimension as $d_{ij}^{(k)} \leftarrow d_{ij}^{(k)} + \text{Uniform}(0, \alpha)$, $\alpha = 0.2$

step3. We then renormalize the representation as $d_{ij} \leftarrow d_{ij} / \sum_{k=1}^{|D_i|} d_{ij}^{(k)}$

Convert T with $n_c + n_d$ columns to $V_{1:n_c,j}, U_{1:n_c,j}, d_{1:N_d,j}$



GAN generates $V_{1:n_c,j}, U_{1:n_c,j}, d_{1:N_d,j}$.

<Reconstruction>

for continuous variables, we reconstruct C_{ij} from U_{ij}, V_{ij}.

$$C_{ij} = 2 U_{ij} \sigma_i^{(k)} + \eta_i^{(k)}, \text{ where } k = \underset{k}{\text{argmax}}\ U_{ij}^{(k)}$$

for categorical features, we simply pick the most probable category as $d_{ij} \leftarrow \underset{k}{\text{argmax}}\ d_{ij}^{(k)}$

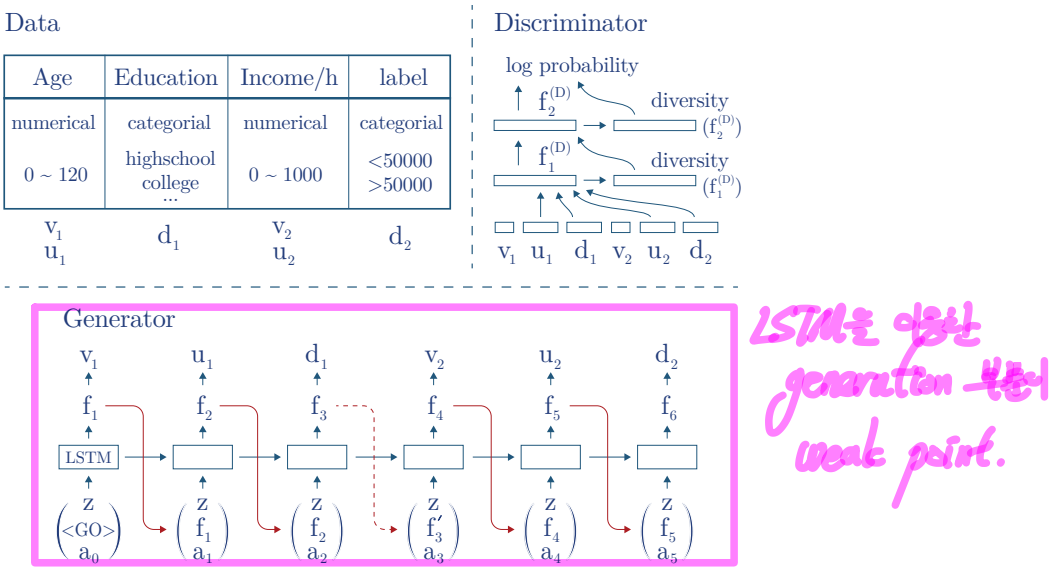


Figure 1: Example of using TGAN to generate a simple census table. The toy example has 2 continuous variables and 2 discrete variables. Our model generates these 4 variables one by one following their original order in the table. Each sample is generated in 6 steps. Each numerical variable is generated in 2 steps while each categorical variable is generated in 1 step. The discriminator concatenates all features together and uses Multi-Layer Perceptron (MLP) to distinguish real and fake data.

and the input of the discriminator in GAN. Note that GAN does not have access to GMM parameters like η and σ.

GAN generates $v_{1:n_c,j}, u_{1:n_c,j}, \mathbf{d}_{1:n_d,j}$. Post-processing is straightforward. For continuous variables, we reconstruct $c_{i,j}$ from $u_{i,j}, v_{i,j}$ as $c_{i,j} = 2v_{i,j}\sigma_i^{(k)} + \eta_i^{(k)}$, where $k = \arg_k \max u_{i,j}^{(k)}$. For categorical features, we simply pick the most probable category as $d_{i,j} \leftarrow \arg\max_k \mathbf{d}_{i,j}^{(k)}$.

3.2 Model and data generation

In GAN, the discriminator D tries to distinguish whether the data is from the real distribution, while the generator G generates synthetic data and tries to fool the discriminator. Figure 1 shows the structure of our TGAN and how to use it to generate tiny tabular data. We use a long-short-term memory (LSTM) network as the generator and use Multi-Layer Perceptron (MLP) in the discriminator.

Generator: We generate a numerical variable in 2 steps. We first generate the value scalar v_i, then generate the cluster vector u_i. We generate categorical feature in 1 step as a probability distribution over all possible labels.

The output and hidden state size of LSTM is n_h. The input to the LSTM in each step t is the random variable z, the previous hidden vector f_{t-1} or an embedding vector f'_{t-1} depending on the type of previous output, and the weighted context vector a_{t-1}. The random variable z has n_z dimensions. Each dimension is sampled from $\mathcal{N}(0, 1)$. The attention-based context vector a_t is a weighted average over all the previous LSTM outputs $h_{1:t}$. So a_t is a n_h-dimensional vector. We learn a attention weight vector $\alpha_t \in \mathbb{R}^t$ and compute context as

$$a_t = \sum_{k=1}^{t} \frac{\exp \alpha_{t,k}}{\sum_j \exp \alpha_{t,j}} h_k. \tag{1}$$

We set $a_0 = 0$. The output of LSTM is h_t and we project the output to a hidden vector $f_t = \tanh(W_h h_t)$, where W_h is a learned parameter in the network. The size of f_t is n_f. We further convert the hidden vector to an output variable.

- If the output is the value part of a continuous variable, we compute the output as $v_i = \tanh(W_t f_t)$. The hidden vector for $t+1$ step is f_t.
- If the output is the cluster part of a continuous variable, we compute the output as $u_i = \text{softmax}(W_t f_t)$. The feature vector for $t+1$ step is f_t.
- If the output is a discrete variable, we compute the output as $\mathbf{d}_i = \text{softmax}(W_t f_t)$. The hidden vector for $t+1$ step is $f'_t = E_i[\arg_k \max \mathbf{d}_i]$, where $E \in \mathbb{R}^{|D_i| \times n_f}$ is an embedding matrix for discrete variable D_i.
- f_0 is a special vector <GO> and we learn it during the training.

Discriminator We use a l-layer fully connected neural network as the discriminator. We concatenate $v_{1:n_c}$, $u_{1:n_c}$ and $\mathbf{d}_{1:n_d}$ together as the input.

We compute the internal layers as

$$f_1^{(D)} = \text{LeakyReLU}(\text{BN}(W_1^{(D)}(v_{1:n_c} \oplus u_{1:n_c} \oplus \mathbf{d}_{1:n_d}))), \tag{2}$$

$$f_i^{(D)} = \text{LeakyReLU}(\text{BN}(W_i^{(D)}(f_{i-1}^{(D)} \oplus \text{diversity}(f_{i-1}^{(D)})))), i = 2:l, \tag{3}$$

where $\oplus$ is the concatenation operation. diversity$(\cdot)$ is the mini-batch discrimination vector [42]. Each dimension of the diversity vector is the total distance between one sample and all other samples in the mini-batch using some learned distance metric. BN$(\cdot)$ is batch normalization, and LeakyReLU$(\cdot)$ is the leaky reflect linear activation function. We further compute the output of discriminator as $W^{(D)}(f_l^{(D)} \oplus \text{diversity}(f_l^{(D)}))$ which is a scalar.

Loss Function The model is differentiable, so we train our model using Adam optimizer [17]. We optimize the generator so that it can fool the discriminator as much as possible. To warm up the model more efficiently, we jointly optimize the KL divergence of discrete variables and the cluster vector of continuous variables by adding them to the loss function. Adding the KL divergence term can also make the model more stable. We optimize generator as

$$\mathcal{L}_G = -\mathbb{E}_{z \sim \mathcal{N}(0,1)} \log D(G(z)) + \sum_{i=1}^{n_c} \text{KL}(u'_i, u_i) + \sum_{i=1}^{n_d} \text{KL}(\mathbf{d}'_i, \mathbf{d}_i), \tag{4}$$

where u'_i and $\mathbf{d}'_i$ are generated data while u_i and $\mathbf{d}_i$ are real data. We optimize the discriminator using conventional cross-entropy loss

$$\mathcal{L}_D = -\mathbb{E}_{v_{1:n_c}, u_{1:n_c}, \mathbf{d}_{1:n_d} \sim \mathbb{P}(\mathbf{T})} \log D(v_{1:n_c}, u_{1:n_c}, \mathbf{d}_{1:n_d}) + \mathbb{E}_{z \sim \mathcal{N}(0,1)} \log D(G(z)). \tag{5}$$

4 Evaluation Setup

In this evaluation, we focus on how well TGAN captures the correlation between variables in the table, and whether data scientists can actually use synthetic data to directly learn models. Synthetic data can benefit data science by enabling data scientists to directly learn models over the synthetic data.

Machine learning efficacy: We first train a TGAN data synthesizer using the real training data $\mathbf{T}$ and generate a synthetic training dataset $\mathbf{T}_{synth}$. We then train machine learning models on both the real and synthetic datasets. We use these trained models on real test data and see how well they perform. Figure 2 shows this process of training and evaluating TGAN.

Does it preserve correlation?: We quantitatively evaluate TGAN's ability to capture correlations between columns by computing the *pairwise* mutual information. We discretize each numeric variable into 20 buckets. We adjust the boundaries of the buckets so that each bucket has around 5% data. We compute the normalized mutual information as

$$\text{NMI}(X, Y) = \frac{1}{\max_{x \in X} \text{E}(x) \times \max_{y \in Y} \text{E}(y)} \sum_{x \in X} \sum_{y \in Y} \mathbb{P}(x, y) \log \frac{\mathbb{P}(x, y)}{\mathbb{P}(x)\mathbb{P}(y)}, \tag{6}$$

where E$(\cdot)$ computes the entropy, and X, Y are two selected columns.

Other data synthesizers: We compare our TGAN synthesizer with three published methods for generating synthetic data.

- GC [34] uses a Gaussian Copula to hierarchically generate multiple synthetic tables in a database.
- BN-Id [35] treats each column independently and learns a Bayesian Network for each column.
- BN-Co [35] uses a Bayes Network to model the correlation between columns, and then samples the data from the learned network.

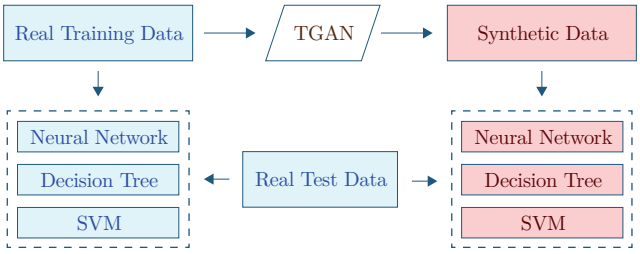


Figure 2: The process of training and evaluating TGAN. The real training data, including the *labels*, is used to learn a GAN and generate synthetic data. Several machine learning models (We choose 5 methods.) are learned using the real training data and the synthetic data. The learned models' accuracy is tested on the real test data that was set aside.

5 Results

Datasets: We selected 3 tabular datasets from the UCI Machine Learning Repository[11]: (1) The Census-Income dataset, which predicts whether the annual income of a person is above or below $50k. (2) The KDD Cup 1999 dataset, which predicts the type of malicious internet traffic encountered. (3) The Covertype dataset, [5] which predicts the forest cover type. Table 1 shows the statistics of the 3 datasets.

Table 1: Dataset statistics. #Features does not count the label column. #Labels is number of the unique labels for classification. #M, #C, #D represent Multi-modal Continuous Feature, Continuous Feature, and Discrete Feature respectively.

Dataset	#Train	#Test	#Features	#Labels	#M/#C	#D
Census	199522	99761	40	2	4/7	33
KDD99	4898431	292300	41	23	22/27	14
Covertype	465589	115423	54	7	1/10	44

How accurate are machine learning models learned on synthetic data? To evaluate the machine learning efficacy of the synthetic data, we follow the process described in Figure 2.

The Census dataset is imbalanced. About 93% of the samples have positive labels, while only 7% have negative labels. Because simply predicting the majority can achieve significantly high accuracy, we used Macro-F1 to evaluate the performance. We selected 5 well-known models: `Decision Tree`, `Linear Support Vector Machine`, `Random Forest`, `AdaBoost`, and `Multi-Layer Perceptron`.

Table 2 shows the Macro-F1 on the Census dataset. We observe that although the Census dataset is simple in a prediction sense, it is challenging in terms of synthetic data generation.

- GC fails to capture the internal relations between columns, so all machine learning models can do nothing but predict the majority.
- BN-Id does not consider the relations between features, so it also fails in training a machine learning model.
- BN-Co is extremely time-consuming in that it is impossible to train it with all the real data. We trained it with 50000 samples, and the maximum degree in the Bayes Network is 2.

Table 2: Macro-F1 evaluation of machine learning models trained on Census dataset. '-' indicates the machine learning model deteriorates into predicting the majority.

Method	Real	GC	BN-Id	BN-Co	TGAN
DT					
`max_depth` $= 10$	74.65	48.61	32.26	32.24	68.70
`max_depth` $= 20$	75.11	48.64	31.16	31.77	64.42
SVM	71.30	-	-	25.69	67.77
RF					
`max_depth` $= 10$, `estimators` $= 10$	59.04	-	-	-	51.42
`max_depth` $= 20$, `estimators` $= 10$	70.95	-	-	32.26	65.89
AdaBoost	74.10	-	-	32.27	70.08
MLP					
`layer_sizes` $= (100,)$	75.47	53.15	25.5	26.34	71.81
`layer_sizes` $= (200, 200)$	73.94	-	-	32.14	68.75

Table 3: Accuracy of machine learning models trained on the real and synthetic training set. (BN-Co fails on KDD99 dataset.)

	KDD99			covertype			
Model	Real	GC	TGAN	Real	GC	BN-Co	TGAN
DT							
`max_depth` $= 10$	97.75	58.34	90.14	77.43	46.10	48.76	69.27
`max_depth` $= 30$	97.35	56.46	80.58	90.82	36.83	46.21	58.88
SVM	93.64	56.15	94.56	70.97	46.30	48.76	67.94
RF							
`max_depth` $= 10$, `estimators` $= 10$	97.79	60.61	93.36	74.58	45.30	48.91	66.60
`max_depth` $= 20$, `estimators` $= 10$	97.81	56.46	92.33	85.13	46.78	48.85	69.33
AdaBoost	19.93	75.94	40.43	49.81	40.95	48.88	66.11
MLP							
`layer_sizes` $= (100,)$	97.48	56.15	95.91	60.32	47.82	48.86	61.24
`layer_sizes` $= (200, 200)$	96.08	56.14	66.38	84.11	46.97	48.79	68.80

- TGAN performs reasonably well. The average performance gap between real data and synthetic data is 5.7% comparing with 24.9% for GC and 43.3% for BN-Co.
- TGAN data in many cases keep the ranking of different machine learning models. This is important because it indicates that a data scientist can evaluate machine learning models on the synthetic dataset and select the best model.

For the KDD99, and Covertype datasets, we use accuracy to compare different methods. Table 3 shows the accuracy of the KDD99 and Covertype datasets. TGAN also consistently outperforms other data synthesizers.

Table 4: Distance between NMI matrices of real data and synthetic data generated by different methods.

	RMSE			MAE		
Dataset	GC	BN-Co	GAN	GC	BN-Co	GAN
Census	0.1474	0.1731	0.0673	0.0657	0.0696	0.0295
KDD99	0.4902	-	0.4551	0.2753	-	0.2476
Covertype	0.4751	0.1252	0.1185	0.2852	0.0256	0.0192

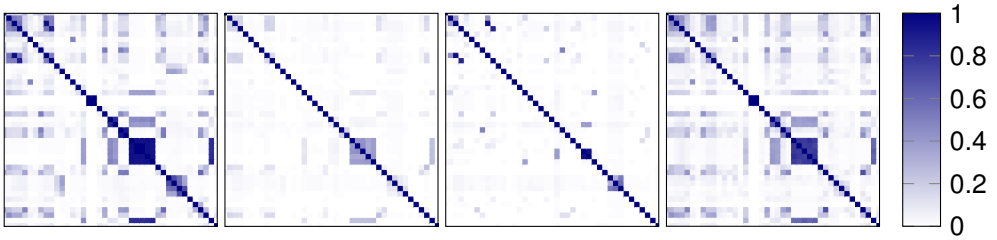


Figure 3: Inspect NMI matrices for Census dataset. From left to right: real data, GC, BN-Co and TGAN.

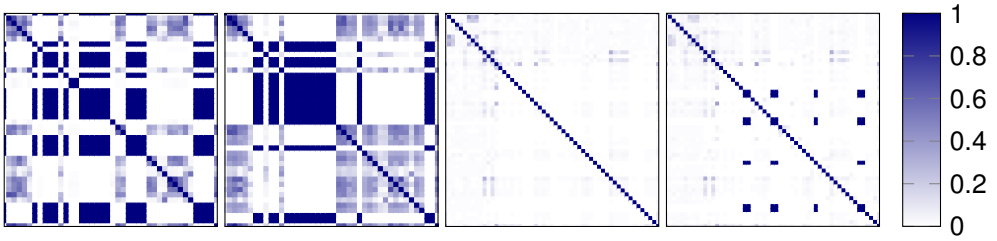


Figure 4: Inspect NMI matrices for KDD99 and Covertype dataset. From left to right: real data (KDD99), TGAN (KDD99), real data (covertype), TGAN (covertype).

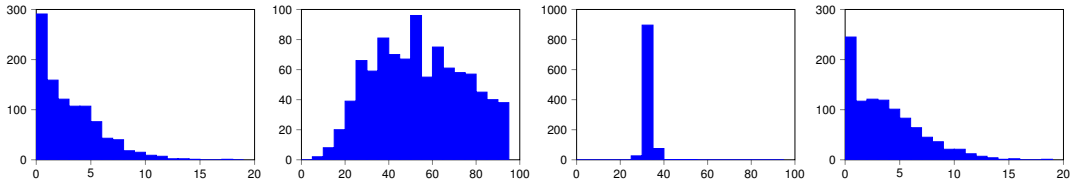


Figure 5: The distribution of the distance to the nearest neighbor on Census dataset. From left to right: $\mathbf{T}'_{real}, \mathbf{T}'_{GC}, \mathbf{T}'_{BN\_Co}, \mathbf{T}'_{TGAN}$

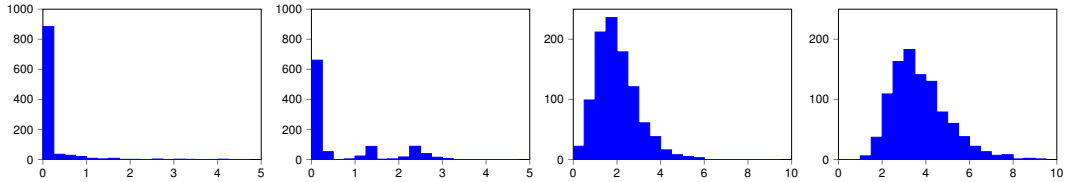
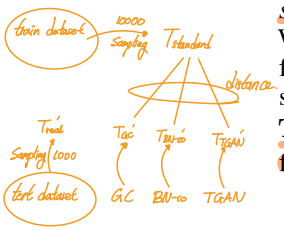


Figure 6: The distribution of the distance to the nearest neighbor on KDD99 and Covertype dataset. From left to right: $\mathbf{T}'_{real}$(KDD99), $\mathbf{T}'_{TGAN}$(KDD99), $\mathbf{T}'_{real}$(Covertype), $\mathbf{T}'_{TGAN}$(Covertype)

Are correlations between variables preserved? Figure 3 visualizes the NMI matrices of real data, GC, BN-Co and TGAN on the Census dataset. Figure 4 visualizes the NMI matrices of real data and TGAN on the KDD99 and Covertype datasets. When compared with GC and BN-Co, TGAN learns the correlation between variables significantly better. Table 4 quantitatively shows the root mean square error and mean absolute error between NMI matrices of real data and synthetic data generated by different methods.



How close is it to real data? Next, we attempt to answer the question: *"Is the data synthesizer simply remembering the data in the training set?"*. To assess this quantitatively we follow these steps. We sample 10000 data points from the training set as $\mathbf{T}_{standard}$. We sample another 1000 data points from the test set $\mathbf{T}'_{real}$. We also sample 1000 data points from different GC, BN-Co, and TGAN synthetic data as $\mathbf{T}'_{GC}, \mathbf{T}'_{BN\_Co}, \mathbf{T}'_{TGAN}$. For data in $\mathbf{T}'$, we compute the distance between $\mathbf{T}'$ and $\mathbf{T}_{standard}$. We compute the distances between $c_{1:n_c,j}, \mathbf{d}_{1:n_d,j}$ from $\mathbf{T}_{standard}$ and $c'_{1:n_c,k}, \mathbf{d}'_{1:n_d,k}$ from $\mathbf{T}'$ as

$$distance(c_{1:n_c,j}, \mathbf{d}_{1:n_d,j}, c'_{1:n_c,k}, \mathbf{d}'_{1:n_d,k}) = \sum_{i=1}^{n_c} \frac{1}{\text{std}(c_i)} |c_{i,j} - c'_{i,k}| + \sum_{i=1}^{n_d} \text{neq}(\mathbf{d}_{i,j}, \mathbf{d}'_{i,k}). \quad (7)$$

Where neq(x, y) is 1 if x and y are different, and 0 otherwise. Figures 5 and 6 show the histogram of nearest neighbor distances on three datasets. The nearest neighbor distance distribution of TGAN is very close to real data.

6 Conclusion

In this paper, we propose TGAN, a GAN-based model for generating relational tables containing continuous and discrete variables. We cluster numerical variables to deal with the multi-modal distribution for continuous features. We add noise and KL divergence into the loss function to effectively generate discrete features. We observe that GANs can effectively capture the correlations between features and are more scalable for large datasets. We show that our model can generate high-quality synthetic data to benefit data science. Relational databases are widely used and very difficult to model. Our model only supports a single table with numerical and categorical features. In the future, we would explore how to model sequential data and how to model multiple tables using GAN.

References

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

[2] Laura Aviñó, Matteo Ruffini, and Ricard Gavaldà. Generating synthetic but plausible healthcare record datasets. *arXiv preprint arXiv:1807.01514*, 2018.

[3] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, and Casey S Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *BioRxiv*, page 159756, 2017.

[4] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[5] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

[6] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. *arXiv preprint arXiv:1807.01202*, 2018.

[7] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

[8] 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 *Data Mining (ICDM), 2017 IEEE International Conference on*, pages 787–792. IEEE, 2017.

[9] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. *arXiv preprint arXiv:1703.06490*, 2017.

[10] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[11] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[12] Qi Dong, Michael R Elliott, and Trivellore E Raghunathan. A nonparametric method to generate synthetic populations to adjust for complex sampling design features. *Survey methodology*, 40 (1):29, 2014.

[13] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[14] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: Better text generation via filling in the _. *arXiv preprint arXiv:1801.07736*, 2018.

[15] Lovedeep Gondara and Ke Wang. Mida: Multiple imputation using denoising autoencoders. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 260–272. Springer, 2018.

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

[17] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[18] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779, 2017.

[19] Bill Howe, Julia Stoyanovich, Haoyue Ping, Bernease Herman, and Matt Gee. Synthetic data for social good. *arXiv preprint arXiv:1710.08874*, 2017.

[20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[21] Kaggle. The state of data science and machine learning 2017, 2017. URL `https://www.kaggle.com/surveys/2017`.

[22] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[23] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jungkwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*, 2017.

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[25] Satkartar K Kinney, Jerome P Reiter, Arnold P Reznek, Javier Miranda, Ron S Jarmin, and John M Abowd. Towards unrestricted public use business microdata: The synthetic longitudinal business database. *International Statistical Review*, 79(3):362–384, 2011.

[26] Matt J Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

[27] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

[28] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[29] Beata Nowok, Gillian M Raab, Chris Dibben, et al. synthpop: Bespoke creation of synthetic data in r. *Journal of statistical software*, 74(11):1–26, 2016.

[30] Junier B Oliva, Avinava Dubey, Andrew G Wilson, Barnabás Póczos, Jeff Schneider, and Eric P Xing. Bayesian nonparametric kernel-learning. In *Artificial Intelligence and Statistics*, pages 1078–1086, 2016.

[31] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10):1071–1083, 2018.

[32] Santiago Pascual, Antonio Bonafonte, and Joan Serra. Segan: Speech enhancement generative adversarial network. *arXiv preprint arXiv:1703.09452*, 2017.

[33] Shreyas Patel, Ashutosh Kakadiya, Maitrey Mehta, Raj Derasari, Rahul Patel, and Ratnik Gandhi. Correlated discrete data generation using adversarial training. *arXiv preprint arXiv:1804.00925*, 2018.

[34] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 399–410. IEEE, 2016.

[35] Haoyue Ping, Julia Stoyanovich, and Bill Howe. Datasynthesizer: Privacy-preserving synthetic datasets. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, page 42. ACM, 2017.

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

[37] Trivellore E Raghunathan, Jerome P Reiter, and Donald B Rubin. Multiple imputation for statistical disclosure limitation. *Journal of official statistics*, 19(1):1, 2003.

[38] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[39] Jerome P Reiter. Simultaneous use of multiple imputation for missing data and disclosure limitation. *Survey Methodology*, 30(2):235–242, 2004.

[40] Jerome P Reiter. Releasing multiply imputed, synthetic public use microdata: An illustration and empirical study. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 168(1):185–205, 2005.

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

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

[43] Isabel Valera, Melanie F Pradier, Maria Lomeli, and Zoubin Ghahramani. General latent feature models for heterogeneous datasets. *arXiv preprint arXiv:1706.03779*, 2017.

[44] Jason Walonoski, Mark Kramer, Joseph Nichols, Andre Quina, Chris Moesel, Dylan Hall, Carlton Duffett, Kudakwashe Dube, Thomas Gallagher, and Scott McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association*, 25(3): 230–238, 2017.

[45] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 515–524. ACM, 2017.

[46] Jingjing Xu, Xu Sun, Xuancheng Ren, Junyang Lin, Binzhen Wei, and Wei Li. Dp-gan: Diversity-promoting generative adversarial network for generating informative and diversified text. *arXiv preprint arXiv:1802.01345*, 2018.

[47] 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. *arXiv preprint arXiv:1712.00164*, 2017.

[48] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. *arXiv preprint*, 2017.

[49] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. Gain: Missing data imputation using generative adversarial nets. *arXiv preprint arXiv:1806.02920*, 2018.

[50] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[51] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.