Permutation-Invariant Tabular Data Synthesis

by

Yujin Zhu

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday August 26, 2022 at 9:00 AM.

Student number:	5235707	
Project duration:	December 1, 2021 – August 26, 2022	
Thesis committee:	Dr. Lydia Chen,	TU Delft, supervisor
	Dr. Rihan Hai,	TU Delft
	Dr. Geethu Joseph,	TU Delft

Abstract

Tabular data synthesis is a promising approach to circumvent strict regulations on data privacy. Although the state-of-the-art tabular data synthesizers, e.g., table-GAN, CTGAN, TVAE, and CTAB-GAN, are effective at generating synthetic tabular data, they are sensitive to column permutations of input data. In this work, we conduct an impact and root-cause analysis of sensitivity to column permutations through extensive empirical analysis. Specifically, we show that changing the input column order increases the statistical difference between real and synthetic data by up to 39%, due to the encoding of tabular data. To address this challenge, we first attempts to find an optimal column order to improve tabular data synthesis. Next, we propose AE-GAN, an effective tabular data synthesizer that leverages the latent representation of tabular data to regulate its sensitivity to column permutations while incurring low training overhead. AE-GAN is composed of an Autoencoder (AE) to efficiently represent tabular data as latent vectors, a Generative Adversarial Network (GAN) to generate realistic synthetic data, and a classifier to improve the semantic integrity of the generated records. It combines the flexibility of unsupervised training with the control offered by supervised training, thereby ensuring the statistical similarity between real and synthetic data. The evaluation of AE-GAN on five datasets shows that it is not only more permutation-invariant than the prior state-of-the-art, but also results in better downstream analysis based on the generated data.

Yujin Zhu
Delft, August 2022

Preface

This project has been a rocky yet fruitful journey for me. Starting with an interest in synthetic data generation, I spent the first few months exploring possible research directions without an outcome. I was discouraged and felt unsure if I could finish this project on time. However, the idea of permutation-invariant tabular data synthesis occurred during a discussion with my supervisor Lydia Chen and daily supervisor Zilong Zhao. After that, this project became on track. In the following months, I immersed myself in experiments and results analysis, through which I learned a lot about experiment design and data analysis. In the end, my work resulted in a submission to the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23).

I would like to thank my supervisor Lydia Chen for your challenging questions, constructive advice, and inspiring talks. I would have learned much less about scientific research without your guidance. I am also grateful to my daily supervisor Zilong Zhao for sharing your knowledge about tabular data synthesis with me. Besides, I want to express my gratitude to Shushu Qin, Oguzhan Ersoy, and Masoud Ghiassi. Your company at the office has made my journey much more enjoyable.

Last but not least, I want to thank my friend Yajun Liu, Ching-chi Chuang, and Joost Verouden for sharing my happy and sad moments in this process. I would not be able to accomplish this project without your constant listening and support.

Yujin Zhu
Delft, August 2020

Contents

1 Introduction

As one of the most common data types, tabular data are ubiquitous in the operation of banks, governments, hospitals, and manufacturers, which lay the foundation of modern society [42]. The synthesis of realistic tabular data, i.e., generating synthetic tabular data that are statistically similar to the original data, is crucial for many applications, such as data augmentation [7], imputation [5, 17], and re-balancing [14, 29, 40]. Another important application is to use generated data to overcome the data sharing restrictions [38] caused by regulations on data protection and privacy, such as European General Data Protection Regulation (GDRP) [15].

Synthesizing realistic tabular data is a non-trivial task. Compared to image and language data, tabular data are heterogeneous - they contain dense continuous features and sparse categorical features. The former can have multiple modes, whereas the latter often have highly-imbalanced distributions [3]. Furthermore, the correlation between features in tabular data is often more complex than the spatial or semantic correlation in image or language data. Related features can be far apart spatially, and multiple features can be inter-correlated.

Although the state-of-the-art tabular data synthesizers show promising results [38, 51, 55], they suffer from a critical and undiscovered limitation: sensitivity to column permutations. Changing the input column order influences the statistical similarity between real and synthetic data, as demonstrated in Figure 1.1. Theoretically, reordering the columns of a table should not change the synthesis quality of a tabular data synthesizer because the position of columns does not imply any semantic information. We call this property column permutation invariance, which is similar to permutation invariance in images [13, 30]. However, our extensive empirical analysis shows that the statistical difference between real and synthetic data increases by up to 39% after changing the input column order. And the main reason is the sparsity issue caused by data encoding.

In this work, we design a novel tabular data synthesizer, AE-GAN, that addresses the limitation of the state-of-the-art, i.e., sensitivity to column permutations. Two key features of AE-GAN are (1) the combination of autoencoder and GAN that reduces the sparsity of encoded

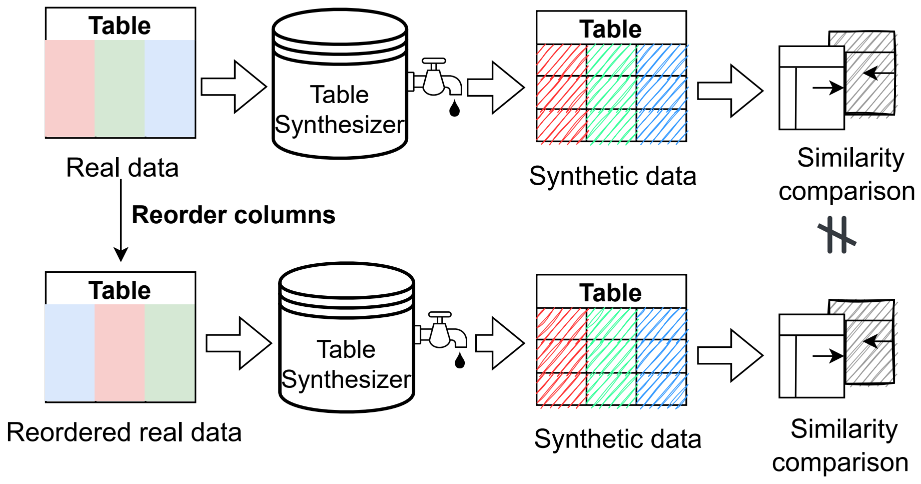


Figure 1.1: Illustration of lacking column permutation invariance of synthetic data, i.e., dissimilarity between real and synthetic data due to reordering of input data.

tabular data and thus enhances column permutation invariance, and (2) the introduction of an auxiliary classifier in Wasserstein GAN with gradient penalty [21] to improve synthesis quality.

We evaluate AE-GAN on five real-world machine learning datasets against four state-of-the-art tabular data synthesizers: table-GAN [38], CTGAN [51], CTAB-GAN[55], and TVAE [51]. The results show that, compared to the baselines, AE-GAN is more permutation-invariant and generates higher quality data leading to more accurate downstream analysis. Moreover, its training time is much shorter than CTAB-GAN, the best performing model in synthesizing realistic tabular data.

This work has three main contributions:

- The first empirical study to analyze column permutation invariance for tabular data synthesizer and reveals its root cause, i.e., data sparsity.
- A novel tabular data synthesizer, AE-GAN, which effectively achieves high permutation invariance and good quality of synthetic data in terms of its machine learning utility.
- A feature sorting algorithm for CNN-based tabular data synthesizers, which helps preserve the relation between highly-correlated features.

The rest of this dissertation is organized as follows: Chapter 2 introduces the theoretical background of synthetic tabular data generation. Chapter 3 discusses the recent development in this field and the limitation of the state-of-the-art, i.e., sensitivity to column permutations. Chapter 4 empirically analyzes the causes of sensitivity to column permutations. Then in chapter 5, the first solution, a feature sorting algorithm for CNN-based tabular data synthesizers, is introduced. Chapter 6 describes the second and main solution AE-GAN, a permutation-invariant tabular data synthesizer. In chapter 7, a thorough experimental evaluation of the proposed solutions is conducted. Finally, chapter 8 concludes this dissertation and discusses the future work.

2 Background

2.1. Synthetic data generation

Synthetic data generation is the creation of artificial data that resemble data collected in real life. Synthetic data preserves the statistical properties of real data and can serve as a supplement/alternative when real data are rare or hard to access, such as healthcare and financial data. Figure 2.1 illustrates the process of synthetic data generation. Real data is the input to a data synthesis model, which then generates synthetic data similar to real data. In the assessment, synthetic data must fulfill the requirements on statistical similarity, and if real data are sensitive, privacy constraints should also be considered.

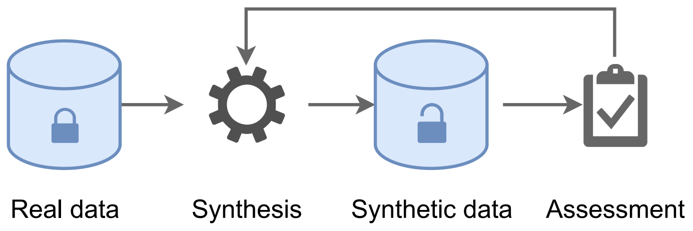


Figure 2.1: Illustration of synthetic data generation

2.2. Generative Adversarial Network

Generative adversarial network (GAN) is a recently developed deep learning algorithm [18] for synthetic data generation. The most basic form is called vanilla GAN. Based on it, many advanced GANs have been brought up to suit various purposes. This section introduces vanilla GAN and two of its most important variations, Wasserstein GAN (WGAN) [1] and Wasserstein GAN with gradient penalty (WGAN-GP) [21].

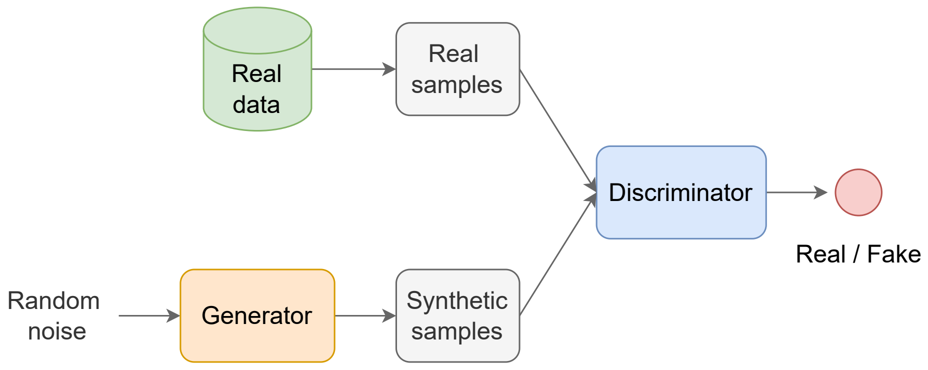


Figure 2.2: The structure of a vanilla GAN

2.2.1. Vanilla GAN

Figure 2.2 shows the structure of a vanilla GAN. It consists of two components: a generator (G) that learns to produce realistic synthetic data, and a discriminator (D) that tries to distinguish between real and synthetic data. Both G and D are neural networks, e.g., Fully-Connected Networks (FCNs) or Convolutional Neural Networks (CNNs).In the training process, G and D play a zero-sum min-max game, described as follows:

$$\underset{G}{min}\ \underset{D}{max}\ V(G,D) = \mathbb{E}[logD(x)]_{x\sim p_{data}(x)} + \mathbb{E}[log(1-D(G(z)))]_{z\sim p(z)}, \tag{2.1}$$

where x is the real sample, $G(z)$ is the synthetic sample generated from input random signal z, and $D(\cdot)$ is the probability of a sample being real from the perspective of D. The goal of G is to minimize the chance that its generated samples are identified as synthetic, whereas D aims at maximizing the chance of correctly identifying real and synthetic samples. Given an optimal D, function 2.1 is equivalent to minimizing the Jensen-Shannon Divergence (JSD) between $p_{data}(x)$ and $p(z)$ [18].

2.2.2. Wasserstein GAN

One of the main limitations of vanilla GAN is the training difficulty. Since G and D play an adversarial game, convergence to an equilibrium is hard. Even if an equilibrium is reached, it is often fleeting. Moreover, it suffers from mode collapse, where G always produces identical samples regardless of its input [14]. Wasserstein GAN is brought up to alleviate those problems. It replaces the JSD used by vanilla GAN with Wasserstein-1 distance (WD), which represents the minimum amount of "mass" to transport from distribution p to distribution q, in order to transform p to q. Given probability distributions $\mathbb{P}_r$ and $\mathbb{P}_g$, the Wasserstein-1 distance is defined as follows:

$$\mathbb{W}(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||], \tag{2.2}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ represents all joint probability distributions $\gamma(x,y)$ whose marginal distributions are $\mathbb{P}_r$ and $\mathbb{P}_g$ [1]. In other words, $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ contains all the transport plans to change $\mathbb{P}_r$ to $\mathbb{P}_g$, and we want to find the greatest lower bound (infimum) of all plans, i.e., the plan with the lowest cost.

The objective of WGAN is:

$$\min_{G} \max_{D} \mathbb{E}[D(x)]_{x \sim p_{data}(x)} - \mathbb{E}[D(G(z))]_{z \sim p(z)}. \tag{2.3}$$

2.2.3. Wasserstein GAN with gradient penalty

Although WGAN has better training stability than vanilla GAN, it has the vanishing gradient problem, where the change in the loss of the generator/discriminator is too small that the network cannot be effectively updated through backward propagation. [21] remedies this problem by adding a gradient penalty term to the loss of D, which leads to the objective of WGAN-GP:

$$\min_{G} \max_{D} \mathbb{E}[D(x)]_{x \sim p_{data}(x)} - \mathbb{E}[D(G(z))]_{z \sim p(z)} + \lambda \mathbb{E}_{\hat{x} \sim p(\hat{x})}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2], \tag{2.4}$$

where λ is the penalty coefficient, $\hat{x} \sim p(\hat{x})$ are the linear interpolations between real and synthetic samples. Due to its training stability and high synthesis quality, WGAN-GP has become the most popular choice in the generation of tables, images and text.

2.3. The representation of tabular data for neural networks

Tabular data, also known as structured data, is a two-dimensional matrix where every row represents a sample, and every column represents a feature. They usually contain both numerical and categorical features, which require different representations.

2.3.1. Categorical features

The representation of categorical features is more challenging than numerical features. In tabular data synthesis, the most frequently used methods are label encoding and one-hot encoding, as explained below:

- Label encoding: It converts a categorical feature to a numerical one by assigning an integer to every category of this feature. Table 2.1 shows an example of label encoding. A categorical feature color has three categories {”red”, ”green”, ”blue”}, and their assigned values are {0, 1, 2} after encoding. The drawback of label encoding is that it introduces an artificial order to the categories, which often leads to sub-optimal results [23]. In our example, ”Blue” > ”Green” > ”Red”, but in real life this order does not exist.

- One-hot encoding: It creates a binary digit for every category of a feature and assigns 1 to the digit that corresponds to the current category, with all other digits set to 0. Table 2.1 also shows an example of one-hot encoding. Since color has 3 categories, its one-hot encoding has 3 digits. From left to right, the 3 digits represent ”Red”, ”Green”, and ”Blue” respectively. If color=”Red”, then the first digit is 1, and the second and third digit are 0. Similarly, the one-hot encoding for ”Green” and ”Blue” are 010 and 001. One clear disadvantage of one-hot encoding is the extra 0s introduced to the data, which will cause memory and scalability issues when many features have a large number of categories.

Table 2.1: Two encoding methods for categorical feature Color

Color	Label encoding	One-hot encoding
Red	0	100
Green	1	010
Blue	2	001

Although there are many approaches for representing categorical features in neural networks (we refer readers to [23] for a comprehensive review), many of them are not suitable for tabular data synthesis due to their irreversibility, i.e., once a value is encoded, it is impossible to restore its original value. Examples include hash encoding, target encoding [35] and leave-one-out encoding [36]. A more advanced approach for representing categorical features is to learn their representations automatically with machine learning models, such as autoencoder (AE) [10] and CNN [22]. It avoids the sparsity issue caused by one-hot encoding and therefore represents tabular data more efficiently. The details of AE is introduced in the following section.

2.3.2. Numerical features

Numerical features include integer and continuous features. Although integer features do not have a continuous distribution, it is usually represented in the same way as continuous features. There are two common methods to represent them:

- Min-max normalization: scale a numerical feature to $[-1, 1]$ using its minimum and maximum values. Given a value v of feature F_i, its normalized value $v_{norm} = -1 + 2 \times \frac{v - min(F_i)}{max(F_i) - min(F_i)}$.
- Mode-specific normalization: this method considers the multiple modes of numerical features, as demonstrated in Figure 2.3. For each continuous feature, it first trains a variational Gaussian mixture (VGM) model with multiple modes, and mode k is a Gaussian distribution with mean η_k and standard deviation ϕ_k. Then, given a value $c_{i,j}$, it identifies the component k that $c_{i,j}$ most likely belongs to, and then normalizes the value by calculating $\alpha_{i,j} = \frac{c_{i,j} - \eta_k}{4\alpha_k}$. The chosen mode $\beta_{i,j}$ is represented by one-hot encoding.

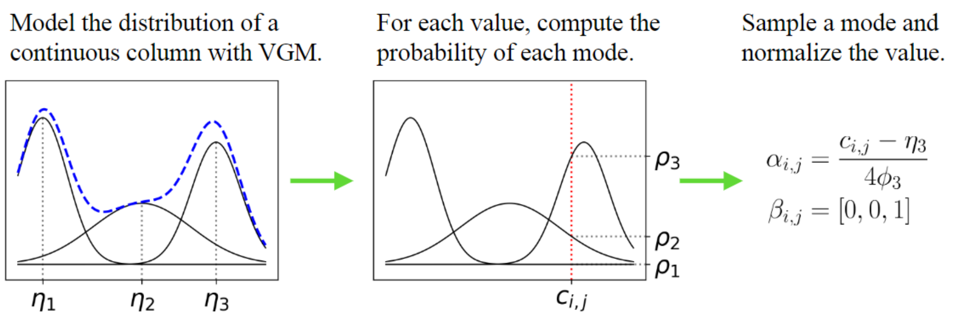

Model the distribution of a continuous column with VGM.
For each value, compute the probability of each mode.
Sample a mode and normalize the value.
η_1
η_2
η_3
$c_{i,j}$
ρ_3
ρ_2
ρ_1
$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$
$\beta_{i,j} = [0, 0, 1]$

Figure 2.3: Illustration of mode-specific normalization [51]

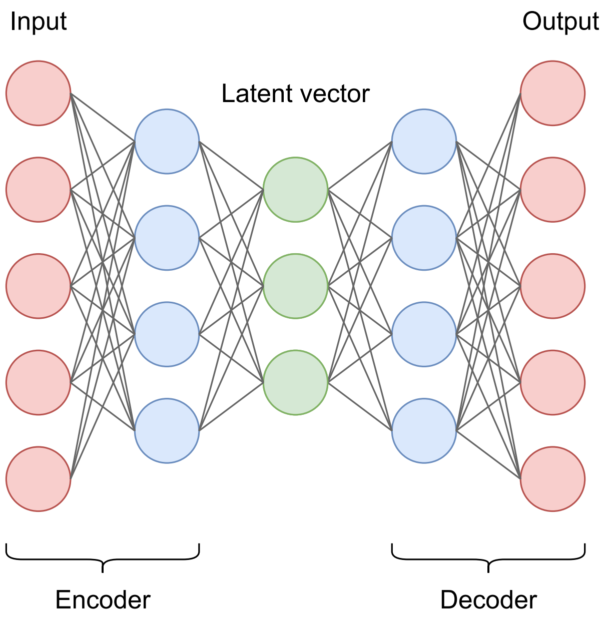


Figure 2.4: The structure of an AE

2.4. Autoencoder

Autoencoder (AE) is an unsupervised learning algorithm that learns a mapping from high-dimensional inputs to low-dimensional representations [37, 47], namely latent vectors. Figure 2.4 shows the structure of an AE. It consists of two FCNs, an encoder (Enc) and a decoder (Dec). Enc takes a high-dimensional input and compresses it to a latent vector, and Dec uses the latent vector to reconstruct the original input. Enc and Dec are penalized for creating output that deviates from the input. The loss function is defined as follows:

$$\min_{\theta,\phi} L(\theta,\phi) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - Dec_\theta(Enc_\phi(x_i))||_2^2, \tag{2.5}$$

where θ and ϕ are the parameters of Dec and Enc, x_i is a high-dimensional input, $Enc_\phi(x_i)$ is the latent vector, $Dec_\theta(Enc_\phi(x_i))$ is the reconstructed output, and N is the total number of samples. The objective of AE is to minimize the difference between its input and output.

3 Related Work

3.1. Tabular data synthesis

We focus on the deep-learning approaches for tabular data synthesis and skip the discussion of classical methods such as Copulas [32, 39] and Bayesian Networks [53]. Table 3.1 summarizes the recently developed deep learning methods for tabular data synthesis, in terms of models, network architecture, and datasets. MedGAN [9] is designed for aggregated electronic health records (EHRs), which only have count and binary features. Since EHRs are high-dimensional and sparse [2], medGAN uses a pre-trained autoencoder to learn compact representations of the input data and thereby simplifies the GAN's task. MedGAN is improved by [2], where the standard GAN loss is replaced by Wasserstein loss with gradient penalty, and the new model is named medWGAN. However, medGAN and medWGAN cannot easily generalize to real-world scenarios because they only consider count and binary features.

A few methods are suitable for general tabular data, including table-GAN [38], CTGAN [51], TVAE [51], and CTAB-GAN [55]. Three of them are based on GAN: table-GAN and CTAB-GAN uses convolutional neural networks as generator and discrminator, whereas CT-GAN's generator and discriminator are fully-connected networks. Besides, table-GAN and CTAB-GAN include a classifier to help training the generator. TVAE is different from the other models because it consists of an encoder and a decoder, both of which are fully-connected networks, and no adversarial training is involved. In addition, CTGAN, TVAE and CTAB-GAN use mode-specific normalization to represent numerical features and one-hot encoding to represent categorical features. In contrast, table-GAN uses label encoding to convert categorical features and then treat it the same as numerical features using min-max normalization. Furthermore, CTAB-GAN defines the mixed datatype, i.e., variables with both continuous and categorical values, and proposes a encoding method for it. Despite their effectiveness in tabular data synthesis, those models overlook and do not abide by the key property of column permutation invariance.

Table 3.1: Deep learning methods for tabular data synthesis (ordered chronologically)

Method	Model design	Network	Data	Data representation
medGAN	AE + GAN	FCN	Medical records	-
table-GAN	DCGAN + Classifier	CNN	General	Min-Max normalization + Label encoding
medWGAN	AE + WGAN-GP	FCN	Medical records	-
CTGAN	Conditional WGAN-GP	FCN	General	Mode-specific normalization + One-hot encoding
TVAE	Conditional VAE	FCN	General	Mode-specific normalization + One-hot encoding
CTAB-GAN	Conditional DCGAN + Classifier	CNN, FCN	General	Mode-specific normalization + One-hot encoding

3.2. Column permutation invariance

Indeed, in the computer vision field, similar concepts have been brought up and investigated, including permutation invariance [13, 30], translation invariance [16, 26, 27], and translation equivalence [11, 12, 49]. Permutation invariance means the output of a neural network stays the same despite permutations of its input. For example, the classification of an image should not change after adjusting the object location in the image.

Motivated by that, we define column permutation invariance in tabular data synthesis as follows: the performance of a tabular data synthesizer is not affected by permutations on the input columns. To the best of our knowledge, column permutation invariance has not been researched by the prior state-of-the-art in tabular data synthesis.

3.3. Synthetic tabular data evaluation

The evaluation of synthetic tabular data involves three parts: statistical similarity, machine learning utility, and privacy [4, 55].

3.3.1. Statistical similarity

Realistic synthetic data should preserve the statistical properties of real data. Therefore, the statistical similarity between real and synthetic data is fundamental in the evaluation of synthetic data. Various metrics can be used to quantify statistical similarity, and the most common ones are the difference in mean and standard deviation, the difference between correlation matrix, Jensen-Shannon divergence, and Wasserstein-1 distance.

3.3.2. Machine learning utility

Another important aspect is the machine learning utility of synthetic data, e.g., whether machine learning models trained with synthetic data can achieve good performance compared to models trained with real data [51]. The evaluation of this property involves several steps:

- Choose a machine learning task and one or several machine learning models for this task.
- Train the models with real samples and test their performance, denoted by P_{real}. The metric can be accuracy or F1 score for classification tasks, or Root Mean Squared Error

or Mean Absolute Error for regression tasks.

- Train the models with synthetic samples and test their performance, denoted by P_{fake}.
- Calculate the difference between P_{real} and P_{fake}. A small difference indicates high machine learning utility.

3.3.3. Privacy

Although privacy protection is one of the main incentives for generating synthetic data, there is no definitive method for evaluating privacy in data synthesis. A common approach is to calculate the distance between each synthetic data point and its closest neighbor in the real dataset and then average over all synthetic data points [38, 50, 55]. More complex methods involve membership attack [44] and differential privacy [31] and are beyond the scope of this work.

4 Empirical Analysis

In this chapter, we show that traditional machine learning models, e.g., logistic regression, decision tree, and support vector machine, are invariant to the column permutations of tabular data. However, the state-of-the-art tabular data synthesizers are sensitive to column permutations. We analyze the root causes of this phenomenon.

4.1. Experiment setup

4.1.1. Dataset overview

The empirical analysis is carried out with five real world datasets, namely Loan, Adult, Credit, Intrusion, and Covtype, as summarized in Table 4.1. The Loan dataset[1] contains the demographic information about bank customers and their response to a personal loan campaign. The Adult dataset[2] has many census data and is used to predict whether the income of an adult exceeds \$50k/year. The Credit dataset[3] consists of anonymized credit card transactions labeled as fraudulent or genuine. The Intrusion dataset[4] has encrypted WiFi traffic records and classifies whether a record is from a unmanned aerial vehicle. The Covtype dataset[5] contains the cover type of forests and the related geographical information. Every dataset has a target column for classification tasks. Due to the limitation of computational resources, we randomly selected 50k samples from the Credit, Intrusion and Covtype datasets.

4.1.2. Column orders

To test whether the state-of-the-art tabular data synthesizers are sensitive to column permutations, three column orders are designed for the datasets:

1. Original order: the column order in the original datasets.

[1]https://www.kaggle.com/code/pritech/bank-personal-loan-modelling/data
[2]https://archive.ics.uci.edu/ml/datasets/Adult
[3]https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
[4]https://archive.ics.uci.edu/ml/datasets/Unmanned+Aerial+Vehicle+%28UAV%29+Intrusion+Detection
[5]https://archive.ics.uci.edu/ml/datasets/Covertype

Table 4.1: Statistics of datasets

Dataset	# Continuous columns	# Categorical columns	# Samples	Target column
Loan	5	8	5k	PersonalLoan
Adult	5	9	48k	Income
Credit	30	1	50k	Class
Intrusion	22	20	50k	Class
Cover_Type	10	45	50k	Cover type

2. Order by type: put all continuous columns on the left side of the table, and all categorical columns on the right.
3. Order by correlation: first calculate the pair-wise correlation between all features, and then sort the features based on the absolute correlation coefficient. Place highly-correlated pairs on the left, and weakly-correlated pairs on the right.

4.1.3. Evaluation metric for column permutation invariance

To quantify permutation invariance, we define Maximum Absolute Variation (MAV) as follows:

$$MAV = max(\mathcal{L}) - min(\mathcal{L}), \tag{4.1}$$

where $\mathcal{L} = \{e_1, e_2 ... e_n\}$ represents the evaluation results of synthetic data in column order $\{1, 2, ..n\}$. Since the scale of the evaluation results varies per dataset, we normalize MAV to allow comparison across different datasets. The normalized MAV is calculated as follows:

$$normalized\ MAV = \frac{max(\mathcal{L}) - min(\mathcal{L})}{min(\mathcal{L})} \tag{4.2}$$

The range of normalized MAV is $[0, +\infty]$. A low normalized MAV means a small difference between the synthetic data in different column orders. Therefore the permutation invariance of the tabular data synthesizer is high.

4.2. Traditional ML models permutation invariance

Intuitively, machine learning models should not be sensitive to the column permutations of tabular data. Take logistic regression as an example. Assume that one feature is the target for regression or classification, and all the other features are the predictors. The formula of logistic regression is as follows:

$$log(\frac{p(X)}{1 - p(X)}) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p, \tag{4.3}$$

where $X = (X_1, ..., X_p)$ are p predictors. In this case, changing the column order is equivalent to altering the locations of $X_1, ..X_p$ in the above formula. Obviously, this does not affect how the model is fitted.

Decision tree, another popular machine learning model, is also invariant to column permu-

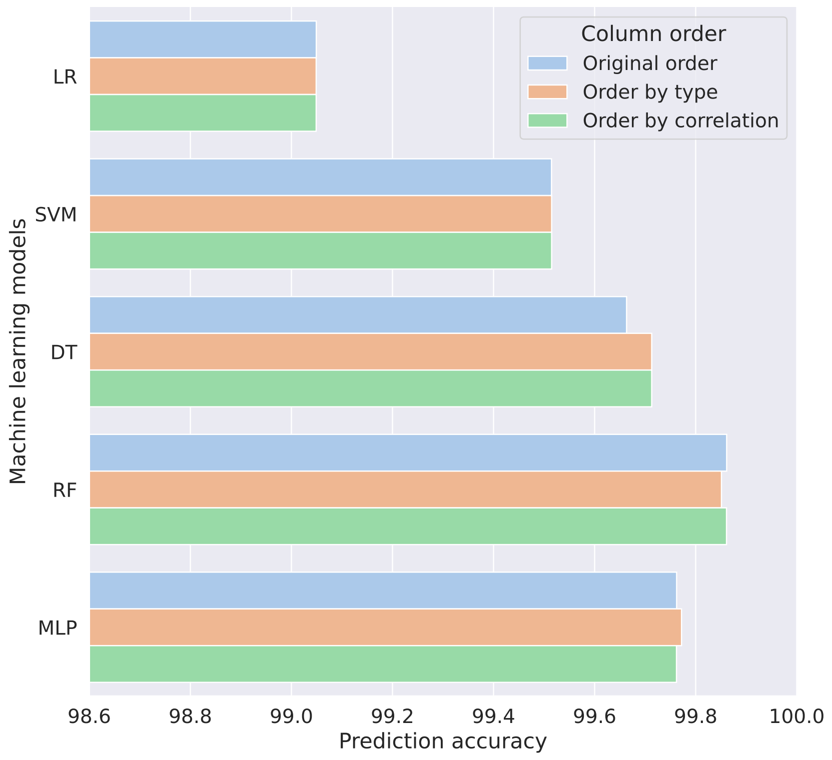


Figure 4.1: Arrange Intrusion dataset in 3 different orders and test the prediction accuracy of 5 machine learning models. The results are almost the same.

tations. In this model, the feature space is divided into multiple regions with linear boundaries. For each division, the feature and cutpoint that minimizes the residual squared error are selected to make the decision boundary. When a tie exists between multiple features or cutpoints, a random choice is made. The order of features does not influence decision trees because it does not affect the selection of features and cutpoints.

To verify that traditional machine learning models are invariant to column permutations, we test 5 machine learning models, logistic regression (LR), support vector machine (SVM), decision tree (DT), random forest (RF), and multi-layer perception (MLP). Both linear and non-linear models are included. We measure the prediction accuracy of the 5 models. Figure 4.1 shows the results on Intrusion dataset. The prediction accuracy of logistic regression and support vector machine stays the same for all column orders, while the prediction accuracies of decision tree, random forest, and multi-layer perceptron change less than 0.2%. The reason is that logistic regression and support vector machine are deterministic algorithms, whereas decision tree, random forest, and multi-layer perceptron are inherently random. Using the formula defined in section 3.2, we find the normalized MAV for the 5 machine learning models are close to 0, which means they are highly invariant to column permutations.

4.3. CNN-based tabular data synthesizers

Initially designed for images, CNNs use a set of convolution kernels to slide over the input feature space, abstract high-dimensional features, and then aggregate them into knowledge about the input. Due to the limited kernel size, CNNs only learn the local relations between neighboring features and fall short to capture the global dependencies.

However, CNNs' focus on local relations hinders tabular data synthesis. In contrast to image data, tabular data do not have strong local relations. Highly-correlated features can be very far apart, and their dependencies are complex and irregular [57]. These characteristics make modeling tabular data extra challenging for CNNs [25, 41], despite their supreme performance in many machine learning tasks [46].

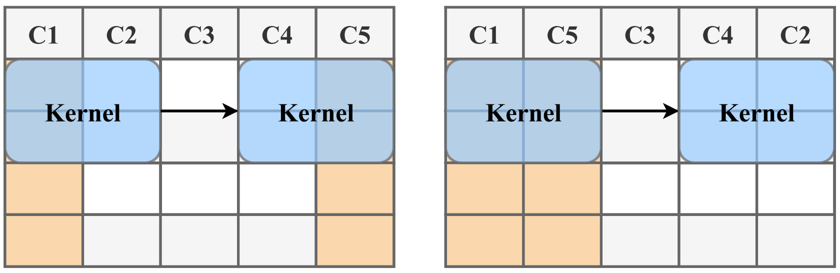


Figure 4.2: Left: CNN cannot capture the dependencies between columns C1 and C5. Right: After exchanging the locations of C2 and C5, C1 and C5 are captured simultaneously [28].

Since CNNs capture mainly local relations, CNN-based tabular data synthesizers are sensitive to column permutations, as demonstrated in Figure 4.2. We use table-GAN to verify this assumption. We test it with five datasets arranged using three column orders, namely the original order, order by type, and order by correlation. For each order, we train the model separately and calculate the Wasserstein-1 distance (WD) between real and synthetic data. Every experiment is repeated 5 times. Table 4.2 summarizes our results. The last column shows the normalized MAV on different datasets. For example, on the Adult dataset, order by type gives the highest WD (12.563), and order by correlation the lowest (11.512). Then the normalized MAV is $(12.563 - 11.512)/11.512 \times 100\% = 9.13\%$. The results show that table-GAN is most sensitive on the Intrusion dataset, with the WD changed by 14.90%.

Table 4.2: TableGAN experiment results using 3 column orders: WD and normalized MAV

Dataset	Column order			Normalized MAV (%)
	Original order	Order by type	Order by corr.	
Loan	2.062	2.047	2.066	0.93%
Adult	12.153	12.563	11.512	9.13%
Credit	0.420	0.410	0.403	4.22%
Covtype	1.282	1.284	1.345	4.91%
Intrusion	6.486	5.896	5.645	14.90%
Avg.	4.481	4.440	4.194	6.82%

4.4. Sparsity v.s. sensitivity

4.4.1. Encoding leads to sparsity

The representation of categorical features in tabular data often leads to sparse training data. In the state-of-the-art, CTGAN, CTAB-GAN, and TVAE use one-hot encoding to represent categorical features. Despite its simplicity and effectiveness, one-hot encoding introduces many 0s to the tabular data and thus increases sparsity. In contrast, table-GAN uses label encoding to transform categorical features into numerical ones and then normalizes them using min-max normalization. Although this method does not increase sparsity, it causes poor synthesis quality of categorical features.

Representing numerical features can also lead to sparsity. CTGAN, TVAE, and CTAB-GAN use mode-specific normalization to represent multi-modal numerical features. However, the multi-model information is stored using one-hot encoding. For example, suppose a numerical feature F has three modes, i.e., its distribution can be decomposed into three Gaussian distributions, denoted by $\mathcal{N}_i(\mu_i, \sigma_i^2), i \in [1,3]$. A value of feature F, v, belongs to mode 1. Then v is represented by vector $[\frac{v-\mu_1}{\sigma_1}, 1, 0, 0]$, where $\frac{v-\mu_1}{\sigma_1}$ is the normalized value of v, and [1, 0, 0] represents mode 1. Therefore, if a numerical feature has n modes, it requires a vector of length $n+1$ to represent one value.

Figure 4.3 shows the number of columns in five datasets before and after using various encoding. It proves that one-hot encoding and mode-specific normalization can significantly increases the number of columns in tabular data. For example, the number of columns in the Credit dataset go from 31 to 301 after mode-specific normalization and one-hot encoding. Naturally, this leads to data sparsity.

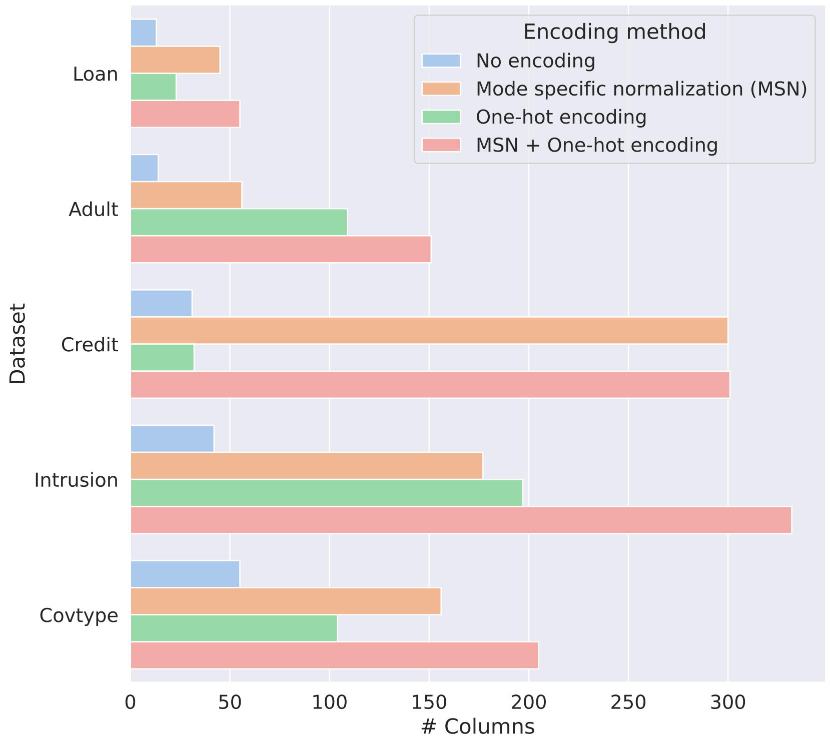


Figure 4.3: The impact of encoding on the numer of columns in five datasets

4.4.2. Sparsity increases sensitivity

Our experiments show that sparse tabular data increases tabular data synthesizers' sensitivity to column permutations. We compare CTAB-GAN, a model using one-hot encoding and mode-specific normalization, with table-GAN, a model using label encoding and min-max normalization. Note that label encoding and min-max normalization do not change the dimensionality of the input data, whereas one-hot encoding and mode-specific normalization increase sparsity. Also recall that table-GAN and CTAB-GAN have a similar architecture: they both use CNNs as the generator and the discriminator. Table 4.3 shows our experiment results of CTAB-GAN. Compared to table-GAN (the results are in Table 4.2), CTAB-GAN is more sensitive to column permutations on all datasets. The average maximum change in WD on five datasets is 38.67%, whereas in table-GAN this is 6.82%.

Table 4.3: CTAB-GAN experiment results using 3 column orders: WD and normalized MAV

Dataset	Column order			Normalized MAV (%)
	Original order	Order by type	Order by corr.	
Loan	0.356	0.283	0.216	64.81%
Adult	1.517	0.934	1.203	62.42%
Credit	0.115	0.144	0.137	25.22%
Covtype	0.539	0.514	0.583	13.42%
Intrusion	2.668	3.401	2.831	27.47%
Avg.	1.039	1.055	0.994	38.67%

In addition, we visualize the input of table-GAN and CTAB-GAN, as shown in Figure 4.4. We reshape each row of a table into a square matrix to make it compatible with CNNs. In table-GAN, one row of the Adult dataset is represented by a 4×4 matrix, but in CTAB-GAN this goes up 24×24 due to the one-hot encoding and mode-specific normalization, and the data points are much sparser. Consequently, CTAB-GAN is much more sensitive to column permutations than table-GAN.

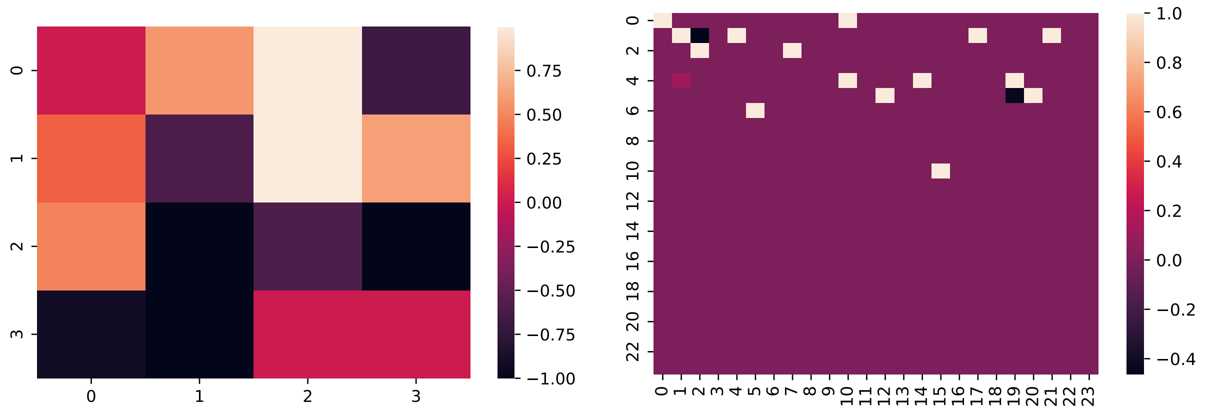


Figure 4.4: Left: Visualization of the input to table-GAN on Adult dataset; Right: Visualization of the input to CTAB-GAN on the Adult dataset.

4.5. FCNs-based tabular data synthesizers

Theoretically, fully-connected networks (FCNs) should be robust to column permutations because all features are connected. However, we find FCNs are not fully permutation invariant while testing CTGAN and TVAE, two FCN-based tabular data synthesizers. Table 4.4 and 4.5 shows our experiment results. Averaging the results on five datasets, the maximum change in WD is 18.62% for CTGAN and 17.29% for TVAE. The results show that FCNs are also sensitive to column permutations.

Table 4.4: CTGAN experiment results using 3 column orders: WD and normalized MAV

Dataset	Column order			Normalized MAV. (%)
	Original order	Order by type	Order by corr.	
Loan	1.271	1.218	0.988	28.64%
Adult	3.762	3.719	3.787	1.83%
Credit	0.255	0.310	0.328	28.63%
Covtype	2.184	2.330	2.047	13.83%
Intrusion	1.814	2.180	1.841	20.18%
Avg.	1.857	1.951	1.798	18.62%

Table 4.5: TVAE experiment results using 3 column orders: WD and normalized MAV

Dataset	Column order			Normalized MAV (%)
	Original order	Order by type	Order by corr.	
Loan	1.550	1.498	1.626	8.54%
Adult	2.598	2.910	3.774	45.27%
Credit	0.221	0.211	0.194	13.92%
Covtype	0.621	0.609	0.633	3.94%
Intrusion	3.627	3.259	3.740	14.76%
Avg.	1.723	1.697	1.993	17.29%

4.6. FCNs v.s. CNNS

We expect FCNs to be more permutation-invariant than CNNs. They are not restricted by the convolution kernel size, and they have the potential to capture global dependencies due to their fully-connected nature. We verify this assumption by comparing the performance of CTGAN and CTAB-GAN.

CTGAN and CTAB-GAN have many similarities, including data representation methods, Wasserstein loss with gradient penalty, and training by sampling. Their main difference lies in the GAN architecture – CTGAN has FCNs, while CTAB-GAN uses CNNs. Therefore, by comparing the performance of CTGAN and CTAB-GAN, we can gain insights into the difference between FCNs and CNNs in terms of column permutation invariance.

Comparing Table 4.3 and Table 4.4, we find CTGAN is less or almost equally sensitive to column permutations on all datasets. Averaging the results on five datasets, its maximum change in WD after column permutation is about 12% lower than the result of CTAB-GAN. Therefore, given the same input, FCNs are more permutation-invariant than CNNs.

5 Feature Sorting Algorithm for CNN-based Tabular Data Synthesizers

5.1. Algorithm design

Although the features in tabular data are often heterogeneous, some are highly correlated, such as age and experience, and height and weight. However, as discussed in section 4.3, CNN-based table synthesizers often fail to capture these correlations due to their focus on local relations. A simple solution for this problem is to put highly-correlated features together, such that a convolution kernel can capture them simultaneously.

The location of features also matters in CNNs. In a convolution process, the features on the border of the input matrix are convoluted fewer times than the features within the border, leading to the potential loss of information about features on the border. This is called boundary effects in image processing [6, 45]. Previous studies have shown that boundary effects result in statistical biases in finite-sampled data [19, 20, 27], and various methods have been proposed to reduce its impact [8, 24, 43]. Therefore, to better capture the correlation between features, one must carefully choose the location of high-correlated features.

Our solution is to put highly-correlated features together in the middle of a table. This brings two advantages: (1) A convolution kernel can capture the highly-correlated features simultaneously so that their correlation is better preserved, and (2) those highly-correlated features will locate at the center of the input matrix, and they are less prone to the boundary effects in CNNs. Figure 5.1 illustrates our idea. Take one row in a table as an example. We reshape it into a square matrix to feed it to a CNN. Before reshaping, two pairs of highly correlated features, $\{F1, F2\}$ and $\{F15, F16\}$, are at the leftmost and rightmost locations. As a result, they are on the border of the input matrix after reshaping. Due to the boundary effects in CNNs, the correlations between $F1$ and $F2$, and $F15$ and $F16$ are probably lost after several aggregations, activation, and pooling operations. However, after putting $\{F1, F2\}$ and $\{F15, F16\}$ in the middle of this row, they end up at the center of the input matrix and are less susceptible to boundary effects.

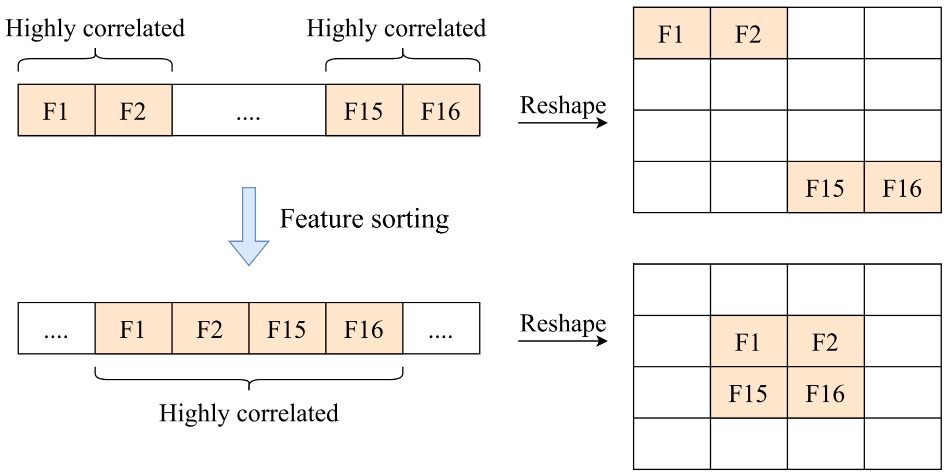


Figure 5.1: Top: The highly correlated features, $F1, F2, F15$ and $F16$ are on the border of the input matrix, suffering from the boundary effects of CNNs. Bottom: After feature sorting, $F1, F2, F15$ and $F16$ are at the center of the input matrix.

Although this idea seems straightforward, one complication arises due to the encoding of features. Some encoding methods, such as Variational Gaussian Mixture [51] and One-hot encoding, require multiple columns to represent one feature. Consequently, each feature may occupy a different number of columns due to encoding. To put the highly correlated features in the middle after encoding, we must consider the length of each encoded feature.

We developed a feature sorting algorithm that groups the highly-correlated features and puts them in the middle, as shown in Algorithm 1. In this algorithm, we first pick out the most correlated pair of features and then add other features to their left or right side. We have two counters, c_{left} and c_{right}, for columns taken by features added to the left and the right side. Before adding a feature, we compare c_{left} and c_{right}, and then add it to the side with fewer columns, such that the highly correlated features always stay in the middle after encoding.

5.2. Algorithm evaluation

We evaluated the feature sorting algorithm on table-GAN and CTAB-GAN, two CNN-based table synthesizers, because this algorithm is designed to alleviate the limitation of CNNs.

5.2.1. Table-GAN

Table 5.1 shows the effect of the feature sorting algorithm on table-GAN. A negative change in Dif. Corr. or WD means the difference between synthetic and real data becomes smaller. That is, the feature sorting algorithm helps table synthesizers generate more realistic data. The results show that the feature sorting algorithm works the best on the Credit dataset, where Dif. Corr. and WD are decreased by 12% and 4%. It also improves the results on the Intrusion dataset, where WD is reduced by 16%, whereas Dif. Corr slightly increases by 3%. However, it doesn't have much influence on the Loan and Covtype datasets, where the Dif.Corr. and WD change less than 5%. Moreover, the results on the Adult dataset become

Algorithm 1 Feature Sorting Algorithm: Put highly-correlated features in the middle

```
Input: Original Table $T_o = \{F_0, F_1, ..., F_n\}$
Output: Sorted Table $T_{sorted}$
 1: $T_{sorted} \leftarrow \{\}$
 2: $c_{left}, c_{right} \leftarrow 0$ {No. columns added to the left / right of $T_{sorted}$}
 3: $corr \leftarrow [\ ]$ {Pair-wise correlation / association}
 4: for all possible pairs of features in $T_o$ do
 5:     Calculate the absolute value of their correlation / association and save it in $corr$
 6: end for
 7: while $length(T_{sorted}) \neq length(T_o)$ do
 8:     Find the largest value $v$ in $corr$
 9:     Find the corresponding pair of features $\{F_x, F_y\}$
10:     $F_{new} \leftarrow \{F_x, F_y\} - \{F_x, F_y\} \cap T_{sorted}$ {Feature(s) not yet in $T_{sorted}$}
11:     $c \leftarrow$ No. columns occupied by $F_{new}$ after encoding
12:     if $T_{sorted}$ is empty then
13:         $T_{sorted} \leftarrow \{F_x, F_y\}$ {Add the first pair}
14:     else
15:         if $c_{right} < c_{left}$ then
16:             $T_{sorted} \leftarrow T_{sorted} + F_{new}$ {Add the new feature(s) to the right}
17:             $c_{right} \leftarrow c_{right} + c$
18:         else
19:             $T_{sorted} \leftarrow F_{new} + T_{sorted}$ {Add the new feature(s) to the left}
20:             $c_{left} \leftarrow c_{left} + c$
21:         end if
22:     end if
23:     Remove $v$ from $corr$
24: end while
25:
26: return $T_{sorted}$
```

worse, with Dif. Corr and WD increased by 24% and 3%.

The algorithm performs the best on the Credit dataset because of the simple correlation between its features. Using ± 0.2 as the threshold for high correlation, only "Time" and "Amount" are strongly-correlated with other features, whereas all the other features have a close-to-0 correlation. Besides, "Time" and "Amount" are only correlated with 3 and 5 features, respectively. With such a small number of correlated features, capturing their relation in the convolution process is easy once we put them together.

The algorithm also alleviates the CNN boundary effect on the highly-correlated features of the Credit dataset. Before applying the algorithm, "Time" and "Amount" are the leftmost and rightmost columns in the table, and many of their correlated features are far apart. However, after feature sorting, these features are put in the middle of the table and therefore become less susceptible to the CNN boundary effect.

In contrast to the Credit dataset, the other four datasets have a larger number of correlated features. For example, in the Adult dataset, most features are correlated with at least one feature, and seven features are correlated with more than three features. Due to the limited kernel size, it is challenging for CNNs to capture all the cross-column relations even after putting the highly-correlated features together. Besides, our algorithm is based on pairwise correlation, but putting one pair of highly-correlated features together could possibly separate another pair of highly-correlated features, which explains why sometimes Dif. Corr. and WD become worse after applying the feature sorting algorithm. In this case, domain knowledge is required to effectively group the correlated features and arrange them in a good order.

Table 5.1: The results of table-GAN before and after applying the feature sorting algorithm on five datasets. The correlation difference and Wasserstein distance between real and synthetic data are reported.

Dataset	Before sorting		After sorting		Change	
	Dif. Corr.	WD	Dif. Corr.	WD	Dif. Corr.	WD
Loan	2.284	2.062	2.203	2.087	-4%	1%
Adult	1.563	12.153	1.942	12.502	24%	3%
Credit	3.092	0.420	2.728	0.403	-12%	-4%
Covtype	4.885	1.282	4.915	1.348	1%	5%
Intrusion	6.433	6.486	6.597	5.418	3%	-16%

5.2.2. CTAB-GAN

To understand whether our feature sorting algorithm works when sparsity is involved, we tested it on CTAB-GAN, and the results are summarized in Table 5.2. Surprisingly, the algorithm can reduce Dif. Corr. and WD by more than 10 % on all datasets except the Credit dataset. On the Loan dataset, the Dif. Corr. and WD are decreased by 57% and 29% after feature sorting, meaning that the algorithm can effectively improve the statistical similarity between synthetic and real data.

Compared with table-GAN, CTAB-GAN has more performance gain after feature sorting. This is due to the sparsity issue caused by the encoding methods of CTAB-GAN, i.e., mode-specific normalization for continuous features and one-hot encoding for categorical features. Since the input data are sparse after encoding, putting the highly-correlated columns together can drastically reduce the distance between correlated columns, and therefore improves CTAB-GAN's ability to capture the relation between highly-correlated columns.

Table 5.2: The results of CTAB-GAN before and after applying the feature sorting algorithm on five datasets. The correlation difference and Wasserstein distance between real and synthetic data are reported.

Dataset	Before sorting		After sorting		Change	
	Dif. Corr.	WD	Dif. Corr.	WD	Dif. Corr.	WD
Loan	1.469	0.356	0.638	0.253	-57%	-29%
Adult	0.448	1.517	0.296	1.205	-34%	-21%
Credit	1.688	0.115	1.660	0.134	-2%	17%
Covtype	1.948	0.539	1.442	0.475	-26%	-12%
Intrusion	3.969	2.668	3.385	1.999	-15%	-25%

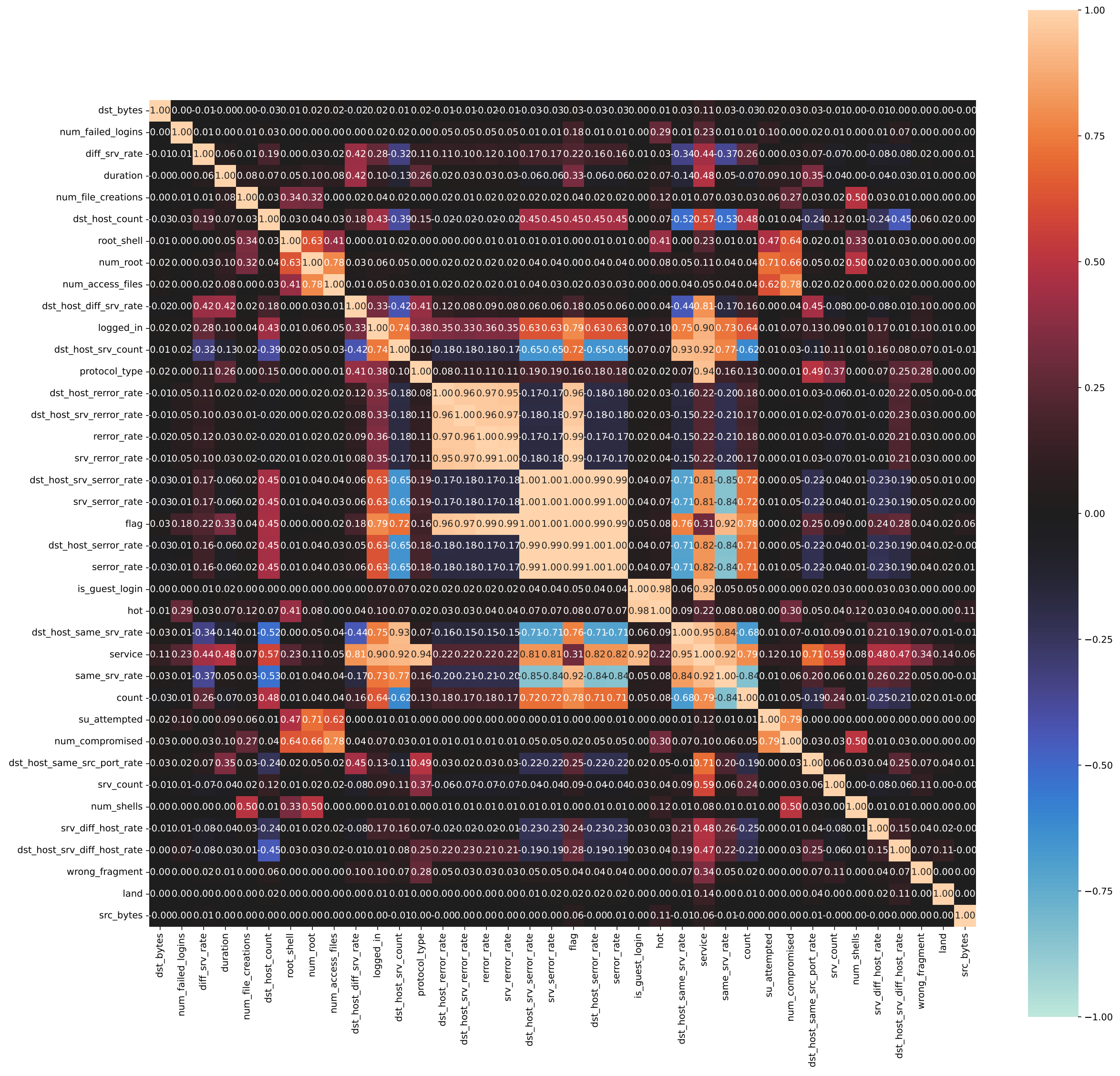


Figure 5.2: The correlation matrix of Intrusion dataset after feature sorting. Light color (orange/blue) means high correlation.

5.2.3. Visualization of feature sorting effect

To verify the effect of the feature sorting algorithm, the correlation matrices of the intrusion dataset before and after feature sorting are plotted, as shown in Figure 5.2 and 5.3. The results show that after feature sorting, the features with high correlation are in the middle of the correlation matrix, meaning that they are put in the middle of all features by the algorithm.

5.2.4. Conclusion

To summarize, our feature sorting algorithm can improve the performance of CNN-based table synthesizers, especially when the input tabular data are sparse. For dense tabular data, it also works if the relation between correlated features is relatively simple. When there are complex relations between many features, finding an optimal column order that improves tabular data synthesis is more challenging.

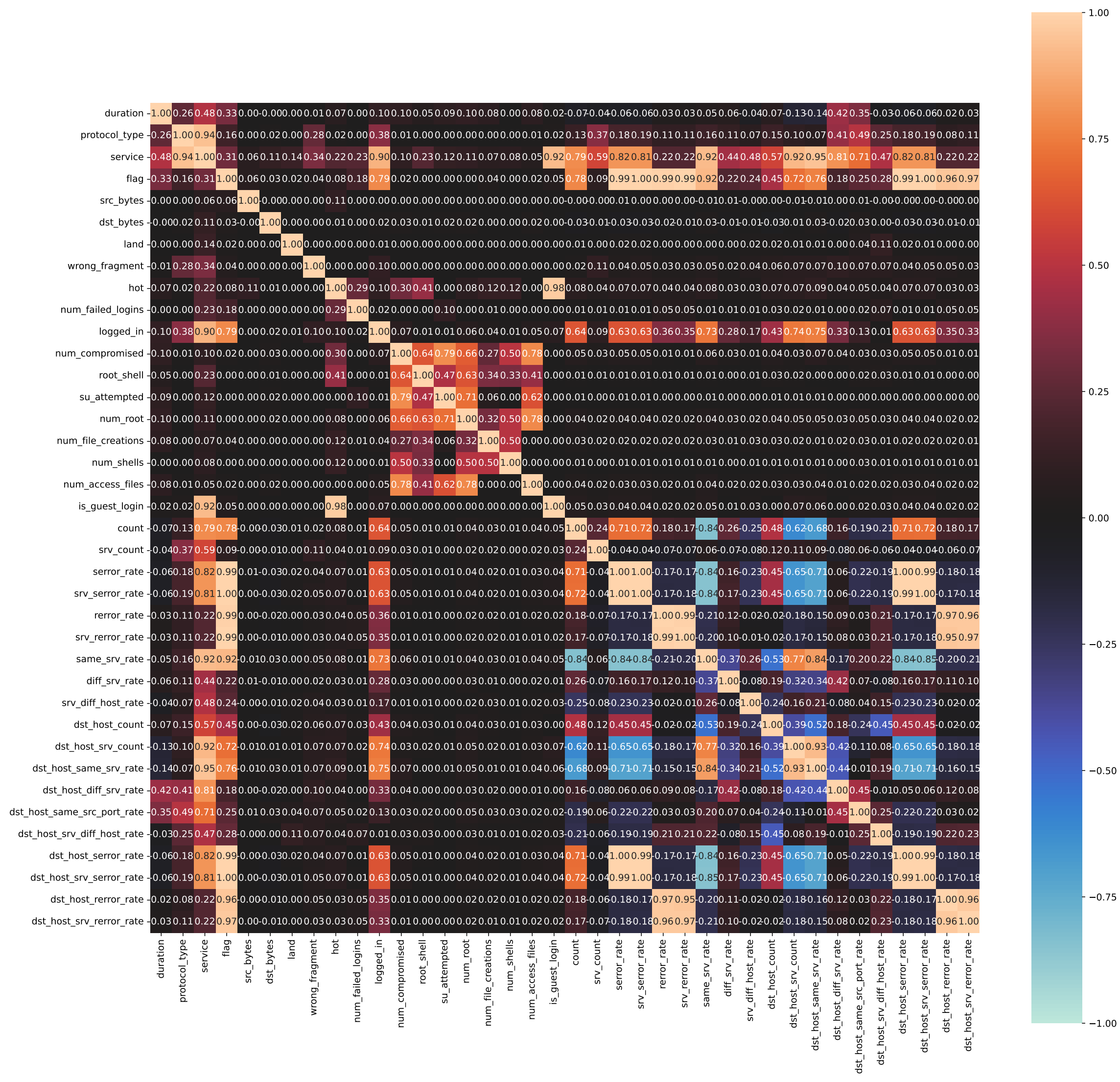


Figure 5.3: The correlation matrix of the Intrusion dataset before feature sorting. Light color (orange/blue) means high correlation.

6 AE-GAN: Permutation-Invariant Tabular Data Synthesizer

We propose AE-GAN, a GAN-based tabular data synthesizer, which is robust to column permutations by utilizing the latent representations of tabular data via an autoencoder. Figure 6.1 shows the overall architecture and the data flow of AE-GAN. It has five components: Encoder (Enc), Decoder (Dec), Generator (G), Discriminator (D), and Classifier (C). The main objective of the encoder and the decoder is to find the latent representation of the input data, which follow the proposed data encoding scheme below. Once the autoencoder is fully trained, the encoded latent vector and the random noise vector are then used as input to train the GAN. Here, the GAN aims to generate the synthetic latent vector with high similarity to the original one. The classifier here provides additional feedback to the training of the GAN, ensuring the semantic integrity of synthetic data. We explain the design choice of each component in the following.

6.1. Data representation

Following [51], we use mode-specific normalization for numerical features and one-hot encoding for categorical features. Mode-specific normalization preserves the multi-modal distribution of continuous variables and improves the performance of tabular data synthesizers [51, 55]. And one-hot encoding is a simple yet effective way to convert categorical values to numerical ones without losing too much information. Certainly, one-hot encoding and mode-specific normalization lead to sparsity, but the autoencoder in AE-GAN solves this issue.

6.2. Encoder and decoder

Since we identify sparsity as one of the main reasons for table synthesizers' sensitivity to column permutations, one natural solution is to use an autoencoder to extract the features of tabular data and compress them into compact latent vectors. Such an advantage can apply to

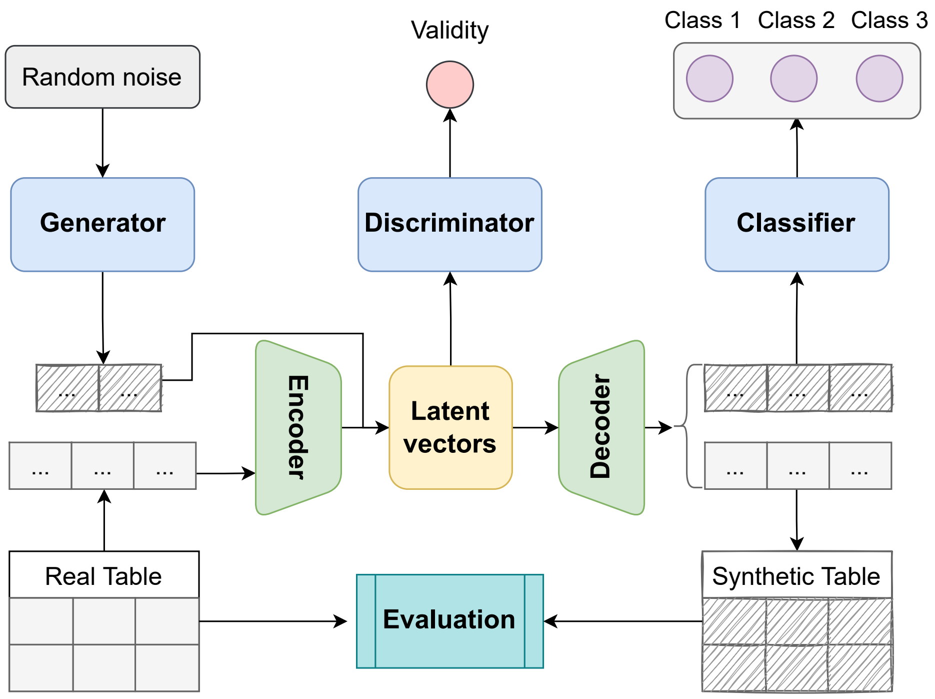


Figure 6.1: The overall architecture and data flow of AE-GAN

all kinds of tabular data synthesizers.

We use two three-layer fully connected networks as Enc and Dec. Based on our study of mainstream open-source implementations of autoencoder [33, 54, 56], fully-connected networks with 2-4 layers are common choices for autoencoders.

Another important design choice is the length of the latent vector, i.e., the output size of Enc and the input size of Dec, which determines the autoencoder's capacity to represent high-dimensional data. We choose this parameter based on the size of the input dataset. For datasets with a large number of columns after encoding, we increase the length of the latent vector to ensure the complex relations between columns are well represented, therefore helping the generator to synthesize realistic data. The latent vector length for each dataset is provided in Table A.2 in Appendix A.

6.3. Generator and discriminator

The core of AE-GAN is a GAN, which has two competing networks, namely the generator, G, and the discriminator, D. We opt for fully-connected networks as the architecture choices for both of them, due to their relative robustness to the column permutation. Figure 6.2 shows the detailed architectures of the generator and the discriminator, both of which have three fully-connected layers. In the discriminator, the fully connected layers are followed by leaky ReLU activation, whereas in the generator they are followed by batch normalization and tanh/leaky ReLU activation. The design is based on the architecture of WGAN-GP [21].

As for the loss function to train both generator and discriminators, we use Wasserstein loss with gradient penalty [21], as it improves the training stability of GAN and alleviates the

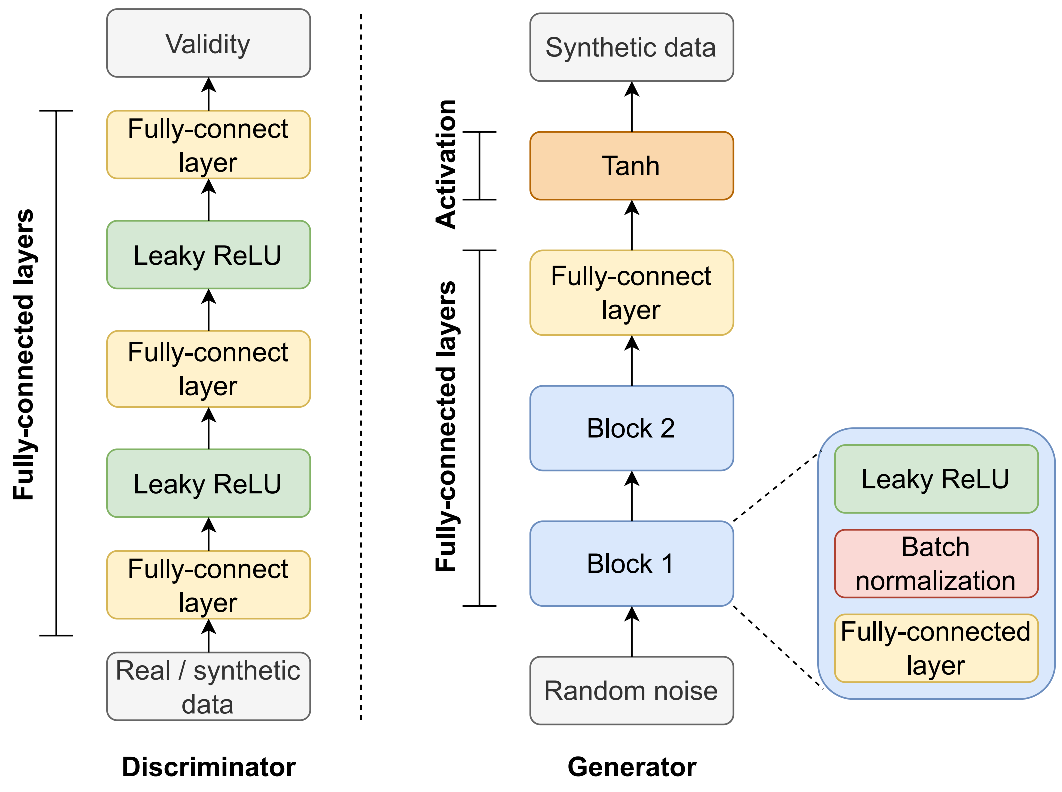


Figure 6.2: Architectures of the discriminator (left) and the generator (right).

vanishing gradient problem.

6.4. Auxiliary classifier

To enhance the training of G and thus the quality of the generated tabular data, we introduce an auxiliary classifier C to the GAN. This design is inspired by [38], where an auxiliary classifier is added to maintain the semantic consistency of synthetic data. Note that the input to C is the reconstructed data from Dec, rather than the latent data from Enc and G because we want C to learn the semantic relations between columns directly. Specifically, given a categorical target column with several categories, C learns to classify which category a sample belongs to according to the columns other than the target column. This is how C differs from the D: D determines the ”realness” of a sample based on all columns in the latent space, whereas C learns the relationship between the target column and all other columns in the reconstructed space. By combining C with the GAN, we simultaneously leverage the flexibility of unsupervised training with the control provided by supervised training, thereby improving the quality of the synthetic data.

Figure 6.3 illustrates the architecture of the auxiliary classifier. It consists of five fully-connected layers, four of which are followed by Leaky ReLU activation and a dropout layers, and the last fully-connect layer predicts the class of a sample.

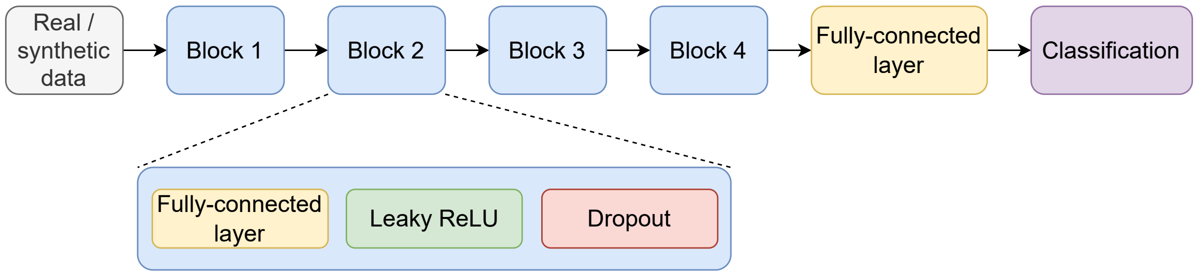


Figure 6.3: Architecture of the auxiliary classifier

6.5. AE-GAN Training

6.5.1. Training loss

The training of AE-GAN requires four loss functions: autoencoder loss $\mathbb{L}_{AE}$, generator loss $\mathbb{L}_G$, discriminator loss $\mathbb{L}_D$, and classifier loss $\mathbb{L}_C$. They are explained in the following.

Autoencoder loss

Autoencoder loss is the reconstruction loss, i.e., the element-wise mean squared error between its input and reconstructed output. It is defined as follows:

$$\mathbb{L}_{AE} = \mathbb{E}||x - \tilde{x}||_2^2, \tag{6.1}$$

where x and $\tilde{x}$ are the input and the reconstructed output.

Generator loss

The generator receives feedback from both the discriminator and the classifier. Therefore, its loss function has two parts: discriminator feedback $\mathbb{L}_G^D$ and classifier feedback $\mathbb{L}_G^C$.

$$\mathbb{L}_G = \mathbb{L}_G^D + \mathbb{L}_G^C \tag{6.2}$$

Discriminator feedback is the validity of the synthetic samples:

$$\mathbb{L}_G^D = -\mathbb{E}[D(G(z))], \tag{6.3}$$

where $G(z)$ is the generator output, i.e., the synthetic sample, and $D(G(z))$ is the discriminator output, i.e., the validity of the sample. The higher $D(G(z))$ is, the more realistic a sample is. Therefore, the generator wants to maximize $D(G(z))$. The minus sign is used to adapt it to a minimization problem.

Classifier feedback is the difference between the predicted labels and the actual labels of synthetic samples. Recall that every synthetic sample has a target column for classification. The classifier predicts the value of the target column based on other columns and then calculates the cross entropy between the predicted value and the actual value of the target column:

$$\mathbb{L}_G^C = H(m, m'), \tag{6.4}$$

where m and m' are the actual and predicted labels of synthetic samples, and $H(\cdot)$ is the cross entropy operator.

Discriminator loss

The discriminator loss measures how well it differentiates the real samples and the synthetic samples. It is calculated by:

$$\mathbb{L}_D = -\mathbb{E}[D(x) - D(G(z)) - \lambda \times (||\nabla D(\hat{x})||_2 - 1)^2], \tag{6.5}$$

where $D(x), D(G(z))$ and $D(\hat{x})$ are the discriminator output on real samples, synthetic samples, and the interpolates between real and synthetic samples. λ is the gradient penalty coefficient, $\nabla D(\hat{x})$ is the gradient of $D(\hat{x})$ on $\hat{x}$.

Classifier loss

The classifier loss also has two parts: loss on real samples $\mathbb{L}_C^R$ and loss on synthetic samples $\mathbb{L}_C^S$.

$$\mathbb{L}_C = \mathbb{L}_C^R + \mathbb{L}_C^S, \tag{6.6}$$

where $\mathbb{L}_C^S = \mathbb{L}_G^C$, and the calculation of $\mathbb{L}_C^R$ is similar to $\mathbb{L}_C^S$.

6.5.2. Training algorithm

We first train the AE until convergence and then train the GAN with the classifier while utilizing the compression power of the AE. Inspired by TimeGAN [52], we also test a joint training algorithm for the AE, the GAN, and the classifier. We first pre-train the autoencoder for a certain number of epochs and then co-train it with the GAN and the classifier. We hypothesize that the joint training of all components can potentially enhance data synthesis because the autoencoder can adjust itself according to the feedback from the GAN and the classifier. This assumption is tested in our experiments in Chapter 7.

7 AE-GAN Evaluation

AE-GAN is evaluated in three aspects: column permutation invariance, synthesis quality, and training time. Our aim is to verify if AE-GAN is robust to column permutations and achieves good synthesis quality compared with state of the art. Additionally, we evaluate the scalability of AE-GAN by analyzing its training time.

7.1. Experimental setup

This section describes the datasets, the evaluation baseline, and the computational environment used for the evaluation.

7.1.1. Datasets

We use five real-world machine learning datasets for the experimental evaluation, namely Loan, Adult, Credit, Intrusion, and Covtype. The details of the datasets are described in Chapter 4.

7.1.2. Baseline

Four state-of-the-art tabular data synthesizers are selected as the baseline models, namely table-GAN, CTGAN, TVAE, and CTAB-GAN. We use the same hyperparameters as the original papers, and every experiment is repeated three times to obtain reliable results.

7.1.3. Computational environment

We implemented our proposed solutions using Pytorch on a server with an Intel(R) Core(TM) i9-10900KF CPU @3.70GHz and a GeForce RTX 2080 Ti GPU.

7.2. Evaluation framework

7.2.1. Metrics

Our evaluation of table synthesizers focuses on the statistical similarity and the machine learning utility difference between real and synthetic data. Specifically, three metrics are adopted, as described below:

- Wasserstein-1 Distance (WD): it measures the difference between two continuous/discrete 1-dimensional distributions. A low WD distance means a high degree of similarity. We use this metric to compare the per feature similarity between real and synthetic data.
- Difference in Correlation Matrix (Dif. Corr.): it measures how well the cross-column correlations are captured by a table synthesizer. A correlation matrix contains the correlation coefficient between every pair of features in a table. We calculate the difference between the correlation matrices of real and synthetic table as follows:

$$Dif.\ Corr. = \sqrt{\sum_{i,j} (Corr^{R}_{i,j} - Corr^{S}_{i,j})^2}, \tag{7.1}$$

  where $Corr^{R}_{i,j}$ and $Corr^{S}_{i,j}$ are the elements at location (i, j) in the real and synthetic correlation matrix.
- Machine learning utility difference: it shows how useful synthetic data are in training machine learning models compared with real data. We measure machine learning utility difference by calculating the difference in accuracy, F1-score, and AUC between machine learning models trained with real and synthetic data. A small difference indicates high synthesis quality.

Note that we use ”correlation” as a general term here. For a pair of continuous features, it refers to the Pearson correlation coefficient; for two categorical features, it is their Cramer's V; and for a categorical feature and a continuous feature, it means their correlation ratio.

7.2.2. Evaluation of column permutation invariance

Similar to our empirical analysis in Chaper 4, we put data in three different orders, i.e., original order, order by type, and order by correlation, and test the column permutation invariance of AE-GAN. We use the normalized MAV as the metric for column permutation invariance, which is defined in section 4.1.3. We demonstrate the evaluation process in Figure 7.1. For each column order, we train a tabular data synthesizer separately. Then we generate synthetic tabular data in each order and calculate their WD to the original data. Next, we find the minimum and maximum values of WD, i.e., WD_{min} and WD_{max}, and the normalized MAV is $\frac{WD_{max} - WD_{min}}{WD_{min}}$. A low normalized MAV indicates high permutation invariance.

7.2.3. Evaluation of synthesis quality

Figure 7.2 shows our evaluation framework for synthesis quality. The real data are divided into train data and test data, of which the train data are used for training the tabular data synthesizer and generating synthetic tabular data. After synthesizing tabular data, we first compare statistical similarity of the train data and synthetic data by calculating their Wasserstein-1 distance and correlation difference. Then we use the train data and synthetic data to fit four

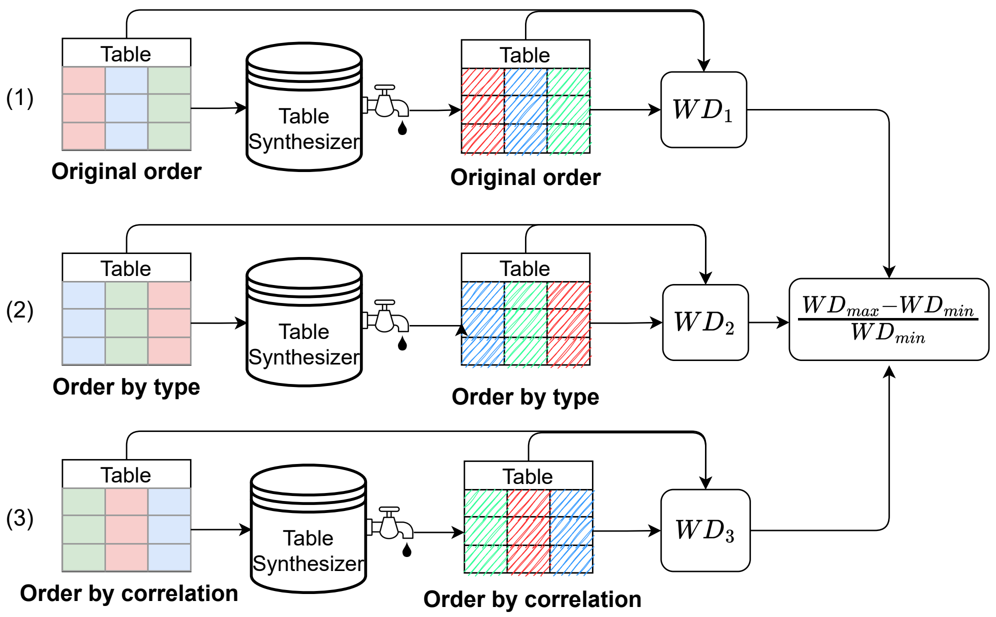


Figure 7.1: Evaluation method for permutation invariance, i.e., the calculation of the normalized MAV

machine learning models separately, and evaluate their accuracy, F1-score and AUC with the test data. By comparing the performance of the machine learning models trained with real and synthetic data, we get their machine learning utility difference.

7.3. AE-GAN column permutation invariance

Table 7.1 shows our evaluation results on the permutation invariance of AE-GAN. The results show that on the Loan, Credit, Covtype, and Intrusion datasets, the normalized MAV of AE-GAN is below or around 10%. However, on the Adult dataset, it is 17.87%. Recall that a low normalized MAV indicates high permutation invariance. Overall, the average normalized MAV on 5 datasets is 10.27%.

To compare the column permutation invariance of AE-GAN against the state-of-the-art tabular data synthesizers, we summarize their evaluation results in Table 7.2. Looking at the results of each dataset, we find the most permutation-invariant model varies per dataset. Our model AE-GAN is the best on the Intrusion dataset, which has the highest number of columns after encoding and therefore is the most difficult dataset. It is also the second best on the Credit dataset, the second largest dataset after encoding.

Comparing the average normalized MAV on five datasets, we find that AE-GAN ranks second in permutation invariance among the five models. Table-GAN is the most permutation-invariant model because it doesn't have the sparsity issue caused by one-hot encoding and mode-specific normalization. However, it has the worst synthesis quality, as our results show later. Although our model AE-GAN is less permutation-invariant than table-GAN, it is better than CTAB-GAN, TVAE, and CTGAN. Those models, like AE-GAN, adopt one-hot encoding for categorical features and mode-specific normalization for numerical features and thus have sparse input data. However, since AE-GAN has an autoencoder to compress the input data,

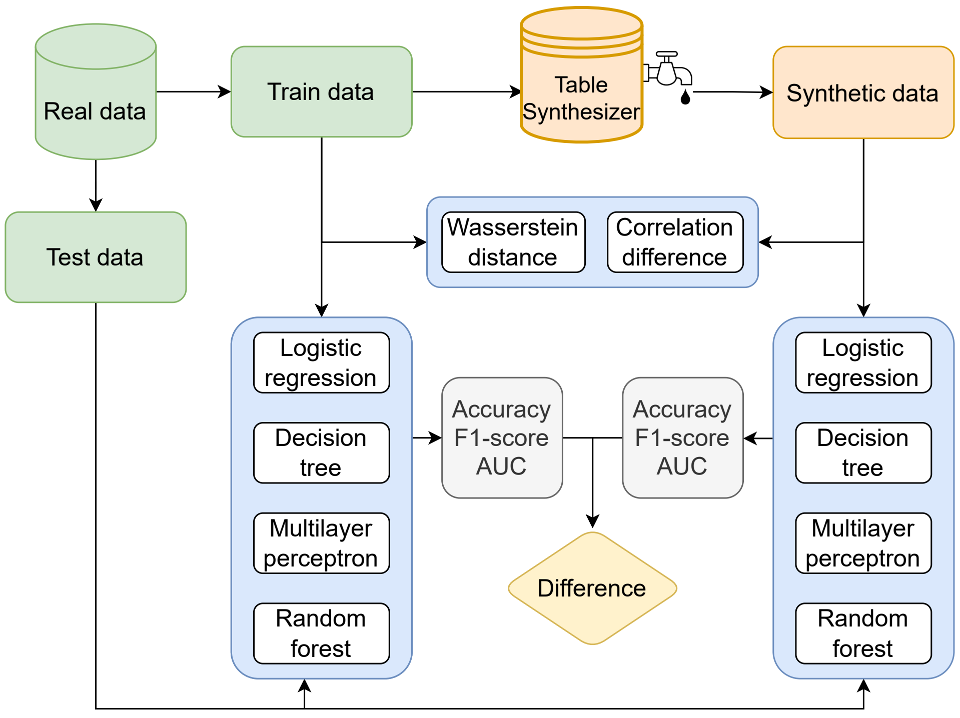


Figure 7.2: Evaluation framework for synthesis quality

it is more robust to column permutations.

Table 7.1: AE-GAN permutation invariance

Dataset	Column order			Normalized MAV (%)
	Original order	Order by type	Order by corr.	
Loan	1.374	1.46	1.474	7.28%
Adult	6.042	5.554	4.962	21.77%
Credit	0.341	0.372	0.348	9.09%
Intrusion	4.328	4.028	4.442	10.28%
Covtype	1.408	1.551	1.414	10.16%
Avg.	2.699	2.593	2.528	11.71%

7.4. AE-GAN synthesis quality

Table 7.3 shows the synthesis quality of AE-GAN and the four baseline models. CTAB-GAN is the best model in terms of statistical similarity and ML utility difference. However, it is the worst in column permutation invariance, as shown in Table 7.2. In contrast, table-GAN is the worst in statistical similarity and ML utility difference, but it is most invariant to column permutations. Among the rest three models, AE-GAN and TVAE do not have a clear advantage over each other. AE-GAN performs better on Dif. Corr and Accuracy, while TVAE has better WD, AUC, and F1-score. Besides, CTGAN performs worse than AE-GAN and TVAE on all metrics except for WD. In short, AE-GAN is better than table-GAN and CTGAN, on par with TVAE, and worse than CTAB-GAN in synthesis quality.

Table 7.2: The normalized MAV of AE-GAN and four baseline models. We test three column orders, namely the original order, order by type, and order by correlation and report the normalized MAV for each dataset. The average normalized MAV on five datasets is also reported.

Model	Dataset					Avg.
	Loan	Adult	Credit	Covtype	Intrusion	
table-GAN	0.93%	9.13%	4.22%	4.91%	14.90%	6.82%
CTGAN	28.64%	1.83%	28.63%	13.83%	20.18%	18.62%
TVAE	8.54%	45.27%	13.92%	3.94%	14.76%	17.29%
CTAB-GAN	64.81%	62.42%	25.22%	13.42%	27.47%	38.67%
AE-GAN	7.28%	21.77%	9.09%	10.16%	10.28%	11.71%

Table 7.3: The statistical similarity and ML utility difference comparison of AE-GAN and four baseline models.

Model	Statistical Similarity		ML Utility Difference		
	WD	Dif. Corr.	Accuracy	AUC	F1-score
table-GAN	4.481	3.651	21.143	0.269	0.387
CTAB-GAN	1.039	1.905	9.114	0.122	0.202
CTGAN	1.857	3.592	14.986	0.219	0.314
TVAE	1.723	2.848	12.839	0.150	0.244
AE-GAN	2.699	2.331	9.984	0.185	0.260

7.5. AE-GAN training time

A model with a short training time can scale up to large datasets. It also requires fewer hardware resources than slow models given the same input. We compare the scalability of AE-GAN with the baseline models by analyzing their training time per epoch. The results are summarized in Table 7.4. Table-GAN and TVAE have the shortest training time, taking 1.58 seconds and 1.40 seconds per epoch on average respectively. AE-GAN and CTGAN are slightly slower than table-GAN and TVAE, using 2.09 seconds and 2.15 seconds per epoch. Surprisingly, CTAB-GAN is more than ten times slower than the other models, taking 25.48 seconds per epoch on average. The results suggest that table-GAN and TVAE have the best scalability. Our model AE-GAN is slower than table-GAN and TVAE due to a more complex model architecture. Nonetheless, it has a clear advantage over CTAB-GAN, which the model with the highest synthesis quality.

Figure 7.3 summarizes the evaluation results of AE-GAN and the baseline model. It shows that, compared with four state-of-the-art models, AE-GAN achieves the best balance between permutation invariance, synthesis quality, and training time. It is more permutation-invariant than CTAB-GAN, CTGAN, and TVAE, and it generates better data than table-GAN and CTGAN. Besides, AE-GAN outperforms CTAB-GAN in training time on all datasets, speeding up as much as 13 times.

Table 7.4: The training time (seconds per epoch) of AE-GAN and the baseline models on five datasets.

Model	Dataset					Avg.
	Loan	Adult	Credit	Intrusion	Covtype	
table-GAN	0.14	0.66	2.33	2.38	2.37	1.58
CTGAN	0.31	1.72	2.55	2.80	3.09	2.09
TVAE	0.09	0.99	2.02	2.43	1.48	1.40
CTAB-GAN	1.62	25.6	28.48	44.97	26.74	25.48
AE-GAN	0.23	2.16	2.18	2.86	3.32	2.15

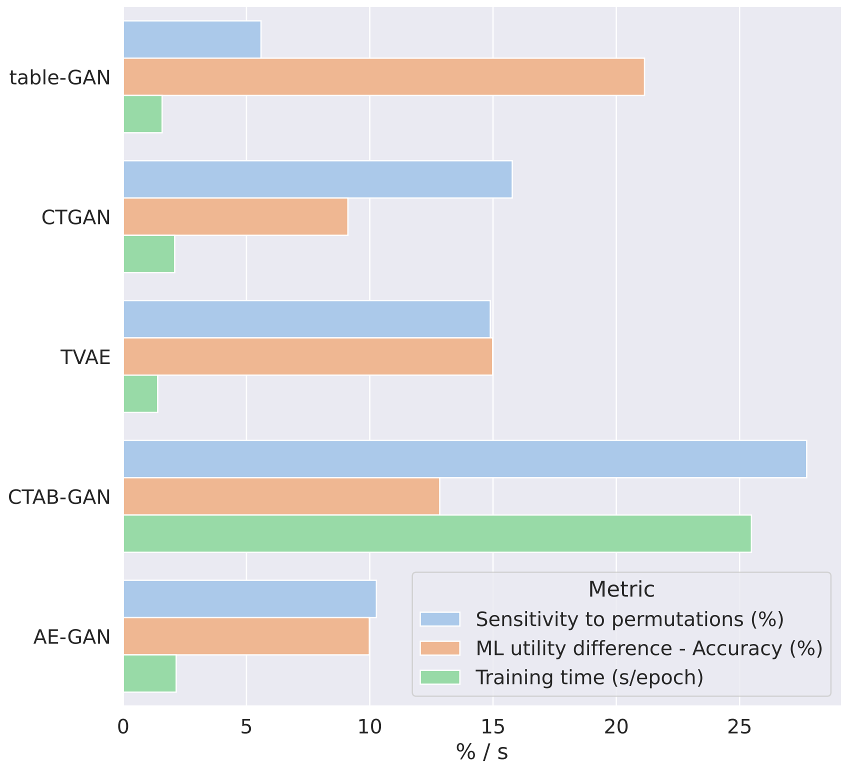


Figure 7.3: Permutation invariance, ML utility difference and training time of AE-GAN and baseline models.

7.6. Ablation study

We did an ablation study to understand the influence of the design choices we made with AE-GAN. We changed the data representations, model architecture, and training algorithm to test their effect. Table 7.5 shows the results of the ablation study.

7.6.1. Without mode-specific normalization (MSN)

We use mode-specific normalization in AE-GAN to normalize numerical features. Although it preserves the multi-model distribution of numerical features, it increases the sparsity in training data. We replace it with min-max normalization to understand its effect. Table 7.5 shows that after removing mode-specific normalization, the WD becomes worse on all datasets, meaning that using mode-specific normalization improves the synthesis quality. However, it also makes AE-GAN less invariant to column permutations. After removing it, the column permutation

Table 7.5: Ablation study results on synthesis quality and permutation invariance of AE-GAN

Dataset	WD between real and synthetic data						Normalized MAV		
	AE-GAN	w/o MSN	w/o one-hot & MSN	w/o classifier	co-train AE & GAN	use information loss	AE-GAN	w/o MSN	w/o one-hot & MSN
Loan	1.374	2.309	3.749	1.308	1.880	1.317	7.28%	3.29%	4.45%
Adult	6.042	6.319	15.432	6.293	6.508	6.357	21.77%	39.20%	40.65%
Credit	0.341	1.650	1.505	0.344	0.534	0.339	9.09%	3.19%	2.70%
Covtype	1.408	3.099	5.954	1.428	2.437	1.439	10.16%	5.48%	0.79%
Intrusion	4.328	14.915	58.241	4.492	4.836	5.487	10.28%	26.84%	2.76%
Avg.	2.699	5.658	16.976	2.773	3.239	2.988	11.71%	15.60%	10.27%

invariance improves on the Loan, Credit, and Covtype datasets. The result suggests that reducing sparsity can improve permutation invariance.

7.6.2. Without one-hot and mode-specific normalization

To further reduce sparsity in the input data, we remove one-hot encoding and mode-specific normalization together. Similar to table-GAN, we pre-process categorical and numerical features using min-max normalization. We found the WD is even worse than only removing mode-specific normalization, which proves that one-hot encoding can also enhance synthesis quality. Furthermore, the permutation invariance improves on all datasets except the Adult dataset. On datasets with many categorical features such as the Covtype and Intrusion datasets, the improvement is around 10%. The results verify again that reducing sparsity can enhance permutation invariance.

7.6.3. Without auxiliary classifier

We use an auxiliary classifier to improve the synthesis quality of AE-GAN. After removing the auxiliary classifier, the Wasserstein distance between real and synthetic data becomes worse on all datasets except the Loan dataset.

7.6.4. Co-training AE and GAN

The AE and GAN in AE-GAN are trained separately. To study whether co-training AE and GAN can improve the synthesis quality, we first pre-train the AE for 300 epochs and then train it together with GAN. Surprisingly, the results show that co-training makes the synthesis quality worse. We find the training loss of AE is already low after pre-training. However, during co-training, the feedback from GAN increases AE's loss and makes it unstable. This suggests that the performance bottleneck of AE-GAN is the GAN, rather than the AE.

7.6.5. Using information loss in GAN

In this paper, we use WGAN-GP and classification loss in GAN. Here we add information loss to the generator loss. Information loss measures the difference in mean and variance between real and synthetic data. Previous study [55] shows that information loss improves the quality of the generated data. In our experiments, we found the information loss only improves synthesis quality on the Loan and Credit datasets.

8 Conclusion

This work investigates permutation-invariant tabular data synthesis. First, an extensive empirical study is conducted to show that the state-of-the-art tabular data synthesizers are sensitive to column permutations. Furthermore, it is observed that the two most commonly used networks in tabular data synthesis, Convolutional Neural Networks (CNNs) and Fully Connected Networks (FCNs), have different levels of sensitivity to column permutations. This work analyzes the root causes of the phenomena. Next, a feature sorting algorithm is deisigned for CNN-based tabular data synthesizers. It improves synthesis quality by putting highly-correlated columns in the middle of the input tabular data, which helps preserve the correlation between different features and alleviates the CNN boundary effect. Then, this work proposes a novel and effective tabular data synthesizer, AE-GAN, that is robust to column permutations and achieves good synthesis quality. The evaluation results show that AE-GAN achieves the best balance between column permutation invariance and synthesis quality. The autoencoder in AE-GAN reduces sparsity in input data, thereby enhancing column permutation invariance. And the auxiliary classifier improves the synthesis quality. Additionally, AE-GAN is much faster than CTAB-GAN, the best state-of-the-art in terms of synthesis quality, demonstrating high scalability.

8.1. Future work

This work focuses on synthesizing tabular data without temporal dependencies. However, tabular time series, i.e., tabular data with features whose data points are collected sequentially over a certain period, have strong temporal dependencies. Therefore, tabular time series not only have correlated features, but each feature also has a correlation between data points, which makes synthesizing tabular time series extra challenging. The current methods for tabular time series synthesis include recurrent neural network [34], convolutional neural network and transformer based on an attention mechanism [48]. A research direction is to evaluate whether those methods are sensitive to column permutations and analyze the reason.

Also, this work concentrates on deep learning methods for tabular data synthesis, namely generative adversarial networks, autoencoder, and variational autoencoder. It is worthwhile to explore statistical methods for tabular data synthesis, such as Copulas [32, 39] and Bayesian Networks [53].

Last but not least, the analysis in this work can be extended to more datasets and column orders, which will shed more light on the column permutation invariance of different tabular data synthesizers. It is left for future work due to the limited time and computational resources.

A
Model hyperparameters

A.1. Training epochs

Table A.1 shows the training epochs of AE-GAN and the four baseline models. The results of baseline models are based on their original implementations.

Table A.1: Training epochs of AE-GAN and the baseline models

Dataset	AE-GAN		TableGAN	CTGAN	TVAE	CTAB-GAN
	AE	GAN				
Loan	300	600	50	300	300	150
Adult	300	150	50	300	300	150
Credit	300	300	50	300	300	150
Covtype	300	300	50	300	300	150
Intrusion	300	300	50	300	300	150

A.2. AE-GAN latent vector length

Table A.2 shows the number of columns in 5 datasets before and after encoding and the latent vector length of AE. The latent vector length increases with the number of columns after encoding.

Table A.2: AE-GAN latent vector length for 5 datasets

Dataset	# Columns w/o encoding	# Columns w encoding	Latent vector length
Loan	13	55	32
Adult	14	151	64
Credit	31	301	96
Intrusion	42	332	96
Covtype	55	205	64

B

Additional experiment results

B.1. AE-GAN Training loss

B.1.1. AE training loss

Figure B.1 shows the training loss of the AE in AE-GAN. It shows that AE performs the worst on the Credit and the Intrusion dataset, which have the longest input vectors after encoding.

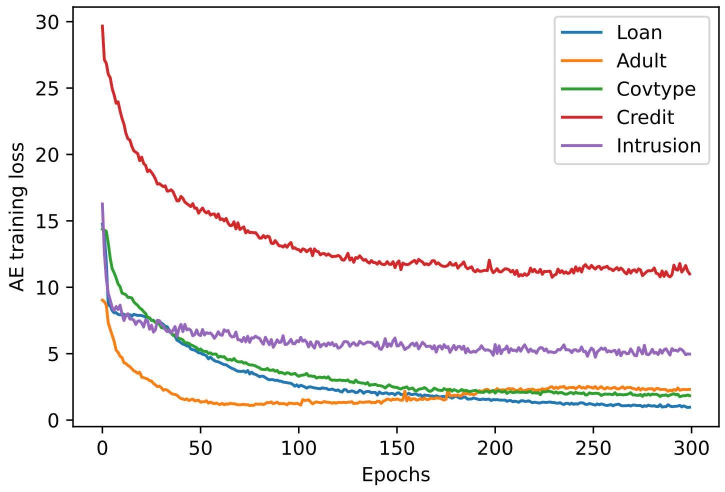


Figure B.1: AE training loss on 5 datasets

B.1.2. GAN training loss

Figure B.2, B.3, and B.4 show the training loss of GAN on Intrusion, Loan, and Covtype datasets. Loss_g is the generator loss, loss_d is the discriminator loss, loss_cc is the classifier loss for real samples, and loss_cg is the classifier loss for synthetic samples. The results show that GAN has converged during training despite some fluctuations in loss_d or loss_cc.

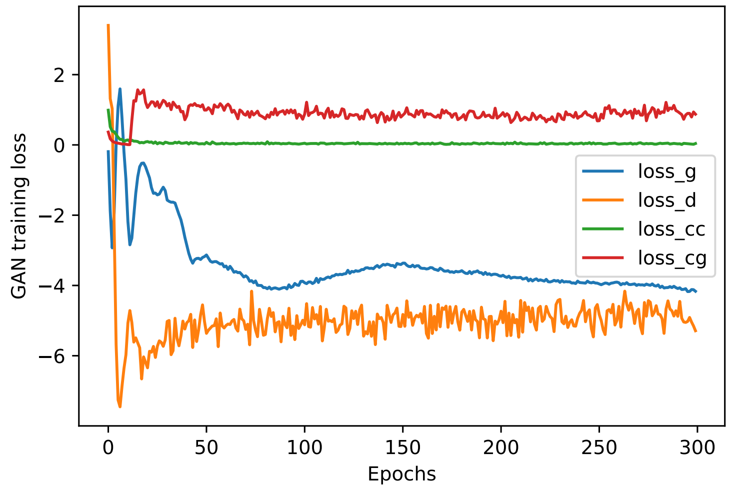


Figure B.2: GAN training loss on the Intrusion dataset.

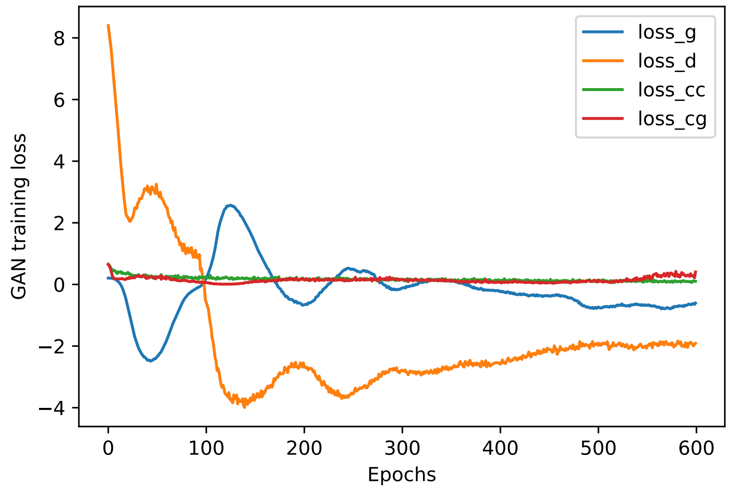


Figure B.3: GAN training loss on the Loan dataset

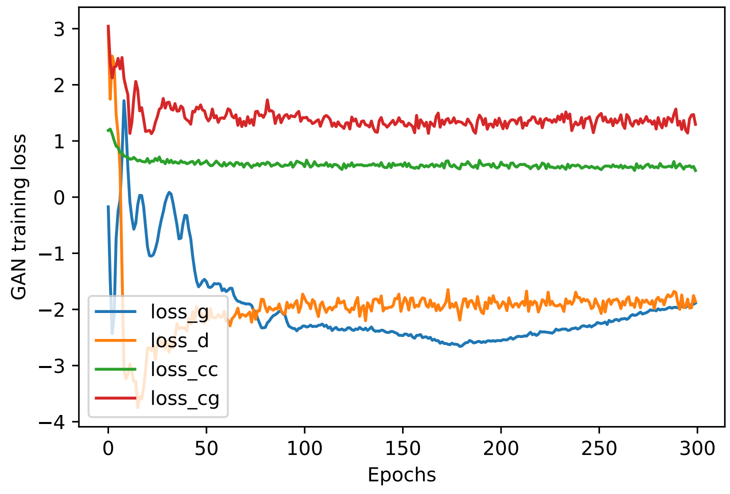


Figure B.4: GAN training loss on the Covtype dataset

B.2. Feature sorting effect

Figure B.5, B.6, B.7, and B.8 show the correlation matrices of the input to table-GAN before and after feature sorting on the Loan and the Adult datasets. The results show that the feature sorting algorithm can effectively put highly correlated features in the middle.

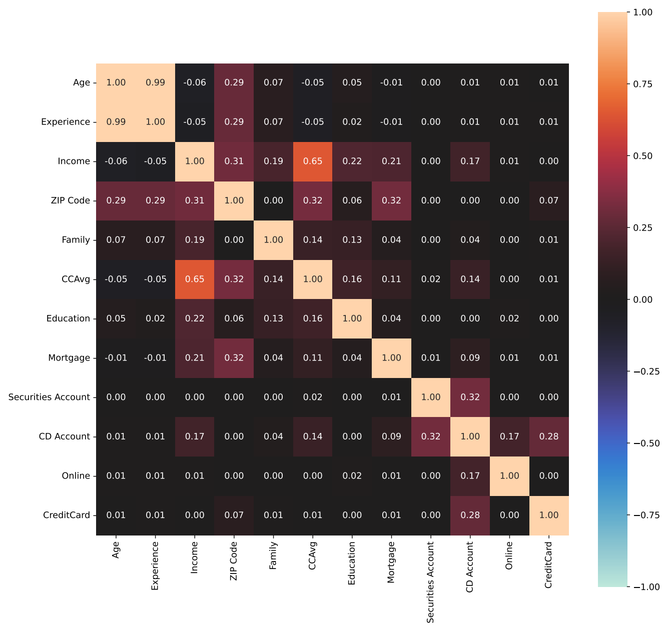


Figure B.5: The correlation matrix of the Loan dataset before feature sorting

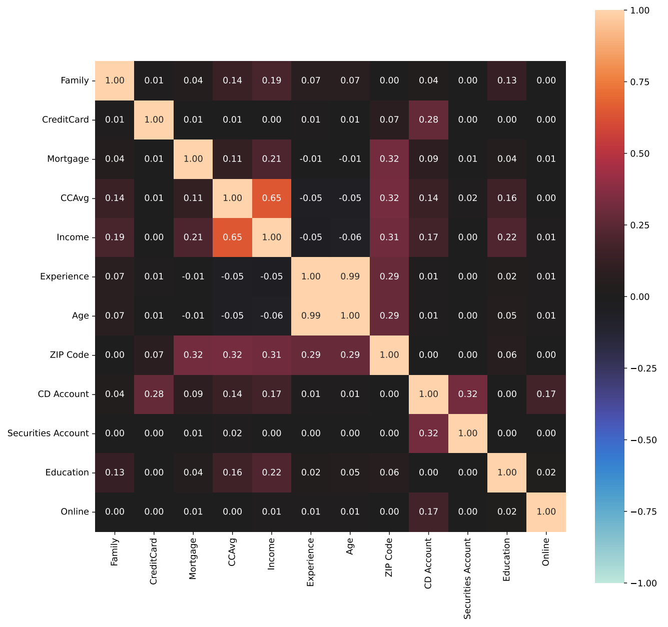
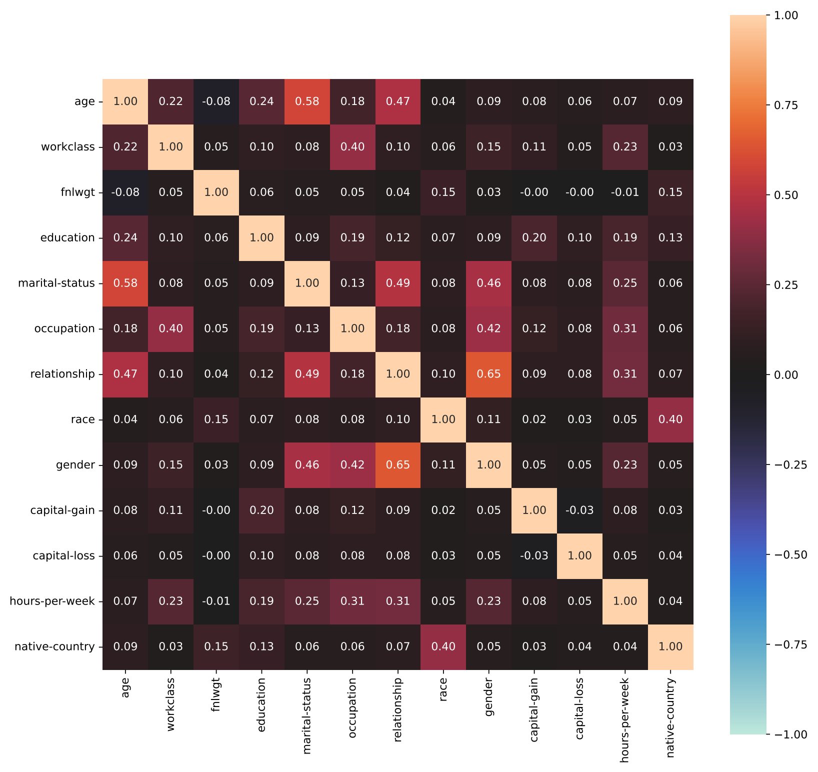
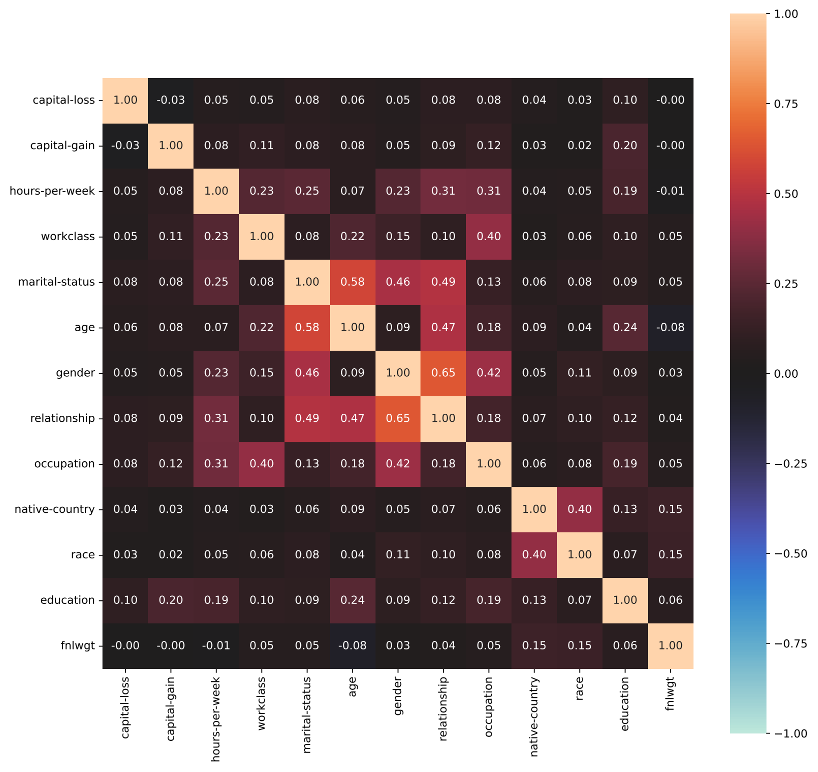


Figure B.6: The correlation matrix of Loan dataset after feature sorting



	age	workclass	fnlwgt	education	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country
age	1.00	0.22	-0.08	0.24	0.58	0.18	0.47	0.04	0.09	0.08	0.06	0.07	0.09
workclass	0.22	1.00	0.05	0.10	0.08	0.40	0.10	0.06	0.15	0.11	0.05	0.23	0.03
fnlwgt	-0.08	0.05	1.00	0.06	0.05	0.05	0.04	0.15	0.03	-0.00	-0.00	-0.01	0.15
education	0.24	0.10	0.06	1.00	0.09	0.19	0.12	0.07	0.09	0.20	0.10	0.19	0.13
marital-status	0.58	0.08	0.05	0.09	1.00	0.13	0.49	0.08	0.46	0.08	0.08	0.25	0.06
occupation	0.18	0.40	0.05	0.19	0.13	1.00	0.18	0.08	0.42	0.12	0.08	0.31	0.06
relationship	0.47	0.10	0.04	0.12	0.49	0.18	1.00	0.10	0.65	0.09	0.08	0.31	0.07
race	0.04	0.06	0.15	0.07	0.08	0.08	0.10	1.00	0.11	0.02	0.03	0.05	0.40
gender	0.09	0.15	0.03	0.09	0.46	0.42	0.65	0.11	1.00	0.05	0.05	0.23	0.05
capital-gain	0.08	0.11	-0.00	0.20	0.08	0.12	0.09	0.02	0.05	1.00	-0.03	0.08	0.03
capital-loss	0.06	0.05	-0.00	0.10	0.08	0.08	0.08	0.03	0.05	-0.03	1.00	0.05	0.04
hours-per-week	0.07	0.23	-0.01	0.19	0.25	0.31	0.31	0.05	0.23	0.08	0.05	1.00	0.04
native-country	0.09	0.03	0.15	0.13	0.06	0.06	0.07	0.40	0.05	0.03	0.04	0.04	1.00

Figure B.7: The correlation matrix of the Adult dataset before feature sorting



	capital-loss	capital-gain	hours-per-week	workclass	marital-status	age	gender	relationship	occupation	native-country	race	education	fnlwgt
capital-loss	1.00	-0.03	0.05	0.05	0.08	0.06	0.05	0.08	0.08	0.04	0.03	0.10	-0.00
capital-gain	-0.03	1.00	0.08	0.11	0.08	0.08	0.05	0.09	0.12	0.03	0.02	0.20	-0.00
hours-per-week	0.05	0.08	1.00	0.23	0.25	0.07	0.23	0.31	0.31	0.04	0.05	0.19	-0.01
workclass	0.05	0.11	0.23	1.00	0.08	0.22	0.15	0.10	0.40	0.03	0.06	0.10	0.05
marital-status	0.08	0.08	0.25	0.08	1.00	0.58	0.46	0.49	0.13	0.06	0.08	0.09	0.05
age	0.06	0.08	0.07	0.22	0.58	1.00	0.09	0.47	0.18	0.09	0.04	0.24	-0.08
gender	0.05	0.05	0.23	0.15	0.46	0.09	1.00	0.65	0.42	0.05	0.11	0.09	0.03
relationship	0.08	0.09	0.31	0.10	0.49	0.47	0.65	1.00	0.18	0.07	0.10	0.12	0.04
occupation	0.08	0.12	0.31	0.40	0.13	0.18	0.42	0.18	1.00	0.06	0.08	0.19	0.05
native-country	0.04	0.03	0.04	0.03	0.06	0.09	0.05	0.07	0.06	1.00	0.40	0.13	0.15
race	0.03	0.02	0.05	0.06	0.08	0.04	0.11	0.10	0.08	0.40	1.00	0.07	0.15
education	0.10	0.20	0.19	0.10	0.09	0.24	0.09	0.12	0.19	0.13	0.07	1.00	0.06
fnlwgt	-0.00	-0.00	-0.01	0.05	0.05	-0.08	0.03	0.04	0.05	0.15	0.15	0.06	1.00

Figure B.8: The correlation matrix of Adult dataset after feature sorting

List of Figures

List of Tables

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 214–223. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/arjovsky17a.html.

[2] Mrinal Kanti Baowaly, Chia-Ching Lin, Chao-Lin Liu, and Kuan-Ta Chen. Synthesizing electronic health records using improved generative adversarial networks. Journal of the American Medical Informatics Association, 26(3):228–241, 2019.

[3] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. CoRR, abs/2110.01889, 2021. URL https://arxiv.org/abs/2110.01889.

[4] Bauke Brenninkmeijer, A de Vries, E Marchiori, and Youri Hille. On the generation and evaluation of tabular data using gans, 2019.

[5] Ramiro Daniel Camino, Christian Hammerschmidt, et al. Working with deep generative models and tabular data imputation. 2020.

[6] Kenneth R Castleman. Digital image processing. Prentice Hall Press, 1996.

[7] Haipeng Chen, Sushil Jajodia, Jing Liu, Noseong Park, Vadim Sokolov, and VS Subrahmanian. Faketables: Using gans to generate functional dependency preserving tables with bounded real data. In IJCAI, pages 2074–2080, 2019.

[8] Hsien-Tzu Cheng, Chun-Hung Chao, Jin-Dong Dong, Hao-Kai Wen, Tyng-Luh Liu, and Min Sun. Cube padding for weakly-supervised saliency prediction in 360 videos. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1420–1429, 2018.

[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. In Machine learning for healthcare conference, pages 286–305. PMLR, 2017.

[10] 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, pages 286–305. PMLR, 2017.

[11] Taco S. Cohen and Max Welling. Steerable cnns. In International Conference on Learning Representations, 2017. URL https://openreview.net/forum?id=rJQKYt5ll.

[12] Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In International Conference on Learning Representations, 2018. URL https://openreview.net/forum?id=Hkbd5xZRb.

[13] Edo Cohen-Karlik, Avichai Ben David, and Amir Globerson. Regularizing towards permutation invariance in recurrent models. Advances in Neural Information Processing Systems, 33:18364–18374, 2020.

[14] Justin Engelmann and Stefan Lessmann. Conditional -based oversampling of tabular data for imbalanced learning. Expert Systems with Applications, 174:114582, 2021.

[15] EU. General data protection regulation, 2018. URL https://gdpr-info.eu/.

[16] Hidetoshi Furukawa. Deep learning for target classification from sar imagery: Data augmentation and translation invariance. arXiv preprint arXiv:1708.07920, 2017.

[17] 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.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014.

[19] D Griffith and Carl Amrhein. An evaluation of correction techniques for boundary effects in spatial statistical analysis: traditional methods. Geographical Analysis, 15(4):352, 1983.

[20] Daniel A Griffith. The boundary value problem in spatial statistical analysis. Journal of regional science, 23(3):377–387, 1983.

[21] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. Advances in neural information processing systems, 30, 2017.

[22] Huimei Han, Ying Li, and Xingquan Zhu. Convolutional neural network learning for generic data classification. Information Sciences, 477:448–465, 2019.

[23] John T Hancock and Taghi M Khoshgoftaar. Survey on categorical data for neural networks. Journal of Big Data, 7(1):1–41, 2020.

[24] Carlo Innamorati, Tobias Ritschel, Tim Weyrich, and Niloy J Mitra. Learning on the edge: Investigating boundary filters in cnns. International Journal of Computer Vision, 128(4):773–782, 2020.

[25] Liran Katzir, Gal Elidan, and Ran El-Yaniv. Net-{dnf}: Effective deep modeling of tabular data. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=73WTGs96kho.

[26] Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. arXiv preprint arXiv:1801.01450, 2017.

[27] Osman Semih Kayhan and Jan C van Gemert. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14274–14285, 2020.

[28] Ethan Keller. Fct-gan: Fourier neural operator for global relation enhancement in tabular data synthesizing using generative adversarial networks, 2022.

[29] Aki Koivu, Mikko Sairanen, Antti Airola, and Tapio Pahikkala. Synthetic minority oversampling of vital statistics data with generative adversarial networks. Journal of the American Medical Informatics Association, 27(11):1667–1674, 2020.

[30] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In International conference on machine learning, pages 3744–3753. PMLR, 2019.

[31] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. Differential privacy: From theory to practice. Synthesis Lectures on Information Security, Privacy, & Trust, 8(4):1–138, 2016.

[32] Zheng Li, Yue Zhao, and Jialin Fu. Sync: A copula based framework for generating synthetic data from aggregated sources. In 2020 International Conference on Data Mining Workshops (ICDMW), pages 571–578. IEEE, 2020.

[33] Xiaoyu Liao. L1aoxingyu/pytorch-beginner: Pytorch tutorial for beginners, 2020. URL https://github.com/L1aoXingyu/pytorch-beginner.git.

[34] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In Proceedings of the ACM Internet Measurement Conference, pages 464–483, 2020.

[35] Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. ACM SIGKDD Explorations Newsletter, 3(1): 27–32, 2001.

[36] Sheraz Naseer and Yasir Saleem. Enhanced network intrusion detection using deep convolutional neural networks. KSII Transactions on Internet and Information Systems (TIIS), 12(10):5159–5178, 2018.

[37] Andrew Ng et al. Sparse autoencoder. CS294A Lecture notes, 72(2011):1–19, 2011.

[38] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. Proc. VLDB Endow., 11(10):1071–1083, jun 2018. ISSN 2150-8097. doi: 10.14778/3231751.3231757. URL https://doi.org/10.14778/3231751.3231757.

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

[40] Matias Quintana and Clayton Miller. Towards class-balancing human comfort datasets with gans. In Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, pages 391–392, 2019.

[41] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 5301–5310. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/rahaman19a.html.

[42] Mark Ryan. Deep learning with structured data. Simon and Schuster, 2020.

[43] Stefan Schubert, Peer Neubert, Johannes Pöschmann, and Peter Protzel. Circular convolutional neural networks for panoramic images and laser data. In 2019 IEEE Intelligent Vehicles Symposium (IV), pages 653–660. IEEE, 2019.

[44] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In 2017 IEEE symposium on security and privacy (SP), pages 3–18. IEEE, 2017.

[45] Gilbert Strang and Truong Nguyen. Wavelets and filter banks. SIAM, 1996.

[46] Farhana Sultana, Abu Sufian, and Paramartha Dutta. Advancements in image classification using convolutional neural network. In 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), pages 122–129. IEEE, 2018.

[47] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. arXiv preprint arXiv:1812.05069, 2018.

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[49] Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 849–858, 2018.

[50] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. arXiv preprint arXiv:1811.11264, 2018.

[51] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling Tabular Data Using Conditional GAN. Curran Associates Inc., Red Hook, NY, USA, 2019.

[52] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. Advances in neural information processing systems, 32, 2019.

[53] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. ACM Transactions on Database Systems (TODS), 42(4):1–41, 2017.

[54] Yue Zhao. Yzhao062/pyod: A comprehensive and scalable python library for outlier detection (anomaly detection), 2020. URL https://github.com/yzhao062/pyod.git.

[55] Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y Chen. Ctab-gan: Effective table data synthesizing. In Asian Conference on Machine Learning, pages 97–112. PMLR, 2021.

[56] Morvan Zhou. Morvanzhou/pytorch-tutorial: Build your neural network easy and fast, 2020. URL https://github.com/MorvanZhou/PyTorch-Tutorial.git.

[57] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A Evrard, James H Doroshow, and Rick L Stevens. Converting tabular data into images for deep learning with convolutional neural networks. Scientific reports, 11 (1):1–11, 2021.