Master Thesis
Data Science

Radboud University

On the Generation and Evaluation of Tabular Data using GANs

Author:
Bauke Brenninkmeijer, BSc
bauke.brenninkmeijer@gmail.com

First supervisor:
prof. dr. ir. A.P. de Vries
a.devries@cs.ru.nl

Second supervisor:
prof. dr. E. Marchiori
elenam@cs.ru.nl

Industry Supervisor:
Youri Hille, MSc
yourihille@gmail.com

December 9, 2019

Abstract

With privacy regulations becoming stricter, the opportunity to apply synthetic data is growing rapidly. Synthetic data can be used in any setting where access to data with personal information is not strictly necessary. However, many require the synthetic data to present the same relations as the original data. Existing statistical models and anonymization tools often have adverse effects on the quality of data for downstream tasks like classification. Deep learning based synthesization techniques like GANs provide solutions for cases where it is vital these relations are kept. Inspired by GANs, we propose an improvement in the state-of-the-art in maintaining these relations in synthetic data. Our proposal includes three contributions. First, we propose the addition of skip connections in the generator, which increases gradient flow and modeling capacity. Second, we propose using the WGAN-GP architecture for training the GAN, which suffers less from mode-collapse and has a more meaningful loss. And finally, we propose a new similarity metric for evaluating synthetic data. This metric better captures different aspects of synthetic data when comparing it to real data. We study the behaviour of our proposed model adaptations against several baseline models on three datasets. Our results show that our proposals improve on the state-of-the-art models, by creating higher quality data. Our evaluation metric captures quality improvements in synthetic data and gives detailed insight into the strengths and weaknesses of evaluated models. We conclude that our proposed adaptations should be used for data synthesis, and our evaluation metric is precise and gives a balanced view of different aspects of the data.

Chapter 1

Acknowledgements

This thesis would not be what it is without the help of my team and supervisor. Specifically, I want to thank Youri Hille, MSc for his continued support and feedback. Additionally, many thanks to Dr. Arjen de Vries and Dr. Masoud Mazloom for their constructive criticism and writing suggestions.

Contents

Chapter 2

Introduction

2.1 Problem and motivation

Ideally, data scientists would use real data for everything. From model training and evaluation to test data for software to plotting the number of sales in a month, real data would be the resource of choice because it captures actual events. However, real data also has many drawbacks such as the need to find, gather and organise it. Additionally, real data is often limited in quantity and might not be sufficient for the requirements of the task. Moreover, real data can be very strictly regulated, for example when it contains personal information.

Synthetic data can give relieve in many of those situations. The main point of data synthesis is generating new, never seen before data. This can be used as a separate dataset for model evaluation and training by data scientists or as test data for software engineers. Because these new datasets do not contain any of the real data points, this method does not let users access the real data, which on its own has many privacy advantages. Furthermore, this data can be used as an addition to existing data. Current deep learning models often require a lot of data for good generalization. Say one has 10,000 rows of some dataset, but for effective generalization one needs ten times more. Realistic synthetic data can then allow training of models that would normally not be available.

However, the way this data is synthesized is changing. Earlier, synthetic data was often created by modeling some joint multivariate probability distribution, which was then sampled. Example models include Bayesian networks and Gaussian Copulas. Some researchers have also found their answers using randomization-based methods [36], but most of these methods have one or more restrictions related to data size or complexity. Recent developments in work on Generative Adversarial Networks (GANs) have shown promising results with tabular data. The question whether GANs would be better than their statistical counterparts rose and several papers have tried to answer [58, 59]. Xu and Veeramachaneni [58] compare GANs on multiple datasets to several statistical methods and show that GANs outperform these classic methods in most tasks.

2.2 Approach

The state of the art for general data synthesis currently resides in the computer vision department, where GANs have taken the world by storm the last five years. The results of these GANs are unmatched by any other technique and seem promising for adaptation in other fields, like tabular data generation. The characteristics of GANs allow it to overcome

issues which earlier methods suffered from, like intractable computations. Instead, the GAN implicitly models the distribution of data to overcome these problems. Intuitively, a GAN can be thought of as someone learning to create fake IDs and bouncers detecting these fake IDs. The person creating the IDs tries to improve their skills and have the IDs look as real as possible, while the bouncer tries to become as good as possible in recognizing the fake IDs.

Due to the flexibility of the GAN framework, many advances have been made in recent years. Regularization methods have been introduced to handle problems with overfitting and mode collapse [5, 23, 45]. Additionally, evaluation metrics specifically for GANs have been proposed, like Inception Score and Frećhet Inception Distance [26]. GANs offer a very diverse and useful framework for many applications, which is why they are securing a lot of spots in top conferences like CVPR and ICLR.

This success has inspired the adoption of GANs for tabular data, such as TGAN[58], TableGAN[50], MedGAN[12] and CTGAN [59]. Where the difficulties with visual data arise from the high dimensionality and complex 2D or 3D relations in the data, tabular data has slightly different problems like having a variety of data types that require different encodings to be suitable for deep learning. In this work, the application of GANs to tabular data is limited to categorical and continuous data. More complex data structures like dates, geospatial data and free text are very interesting directions of research for future work.

2.3 Contributions

In this thesis we propose three contributions. The first and second contributions are changes to the current state of the art model: TGAN [58]. The third contribution is proposing a general synthesized tabular data evaluation method. Using this method, we also present the first apples to apples comparison between current State-of-the-Art methods [1].

2.3.1 Improve TGAN using skip-connections

The first contribution is to use the WGAN-GP training method instead to the original GAN method with adapted loss function. This training method has shown considerable improvements with regards to mode collapse and sample diversity in images [5, 23, 43]. This approach is detailed in 5.1.2.

2.3.2 Improve TGAN using the WGAN-GP architecture

The second addition is the inclusion of skip connections to the generator. This addition has revolutionized discriminative models, leading to well known architectures like ResNet [24] and DenseNet [28], and the hypothesis is this addition will improve the approximation of the real data when used in GANs. MedGAN [12] already shows improvements using this technique in their generators. This approach is detailed in section 5.1.1..

2.3.3 New metric for evaluating synthetic data: Similarity Score

So far, literature has been somewhat disappointing in their evaluations. To name a few issues, the baselines are often quite weak and have not been other State-of-the-Art generation

[1] The paper on CTGAN [59] does this partly as well. Their results were published in July 1. I became aware of their results on July 5.

methods, but rather models like Variational Autoencoders (VAE) [35] or data perturbation tools [1, 57]. Additionally, the evaluation is done with a range of metrics and different datasets, which makes comparing them quite hard. Papers often evaluate using one or more of the following three metrics: privacy, statistics and machine learning.

Privacy evaluations range from membership attacks to information leakage. Statistical methods include correlations and standard deviations. Machine learning capabilities cover the usability of data with regards to model training and evaluation.

In this work I propose a new method that combines these three aspects called the **similarity score**. Of the three mentioned aspects, the similarity score has a focus on the statistical and machine learning ones. Evaluation metrics used by the similarity score include regressions on what we call association matrices and the F1-scores/RMSE-scores of datasets on different classifiers. Our evaluation library is available as a standalone package for python[2].

[2]Available on `https://github.com/Baukebrenninkmeijer/TableEvaluator` and `https://pypi.org/project/table-evaluator`

Chapter 3

Preliminaries

To understand our contributions, several concepts related to machine learning need to be clear. Here, we elaborate on some core concepts from machine learning, and more specifically Generative Adversarial Nets (GANs) to allow for a better understanding of our approach.

3.1 Synthetic Data

Synthetic data can be created in three ways. One, by using some perturbation of real data. Two, by combining attributes from the real data and three, by generating samples from some distribution. All three methods have advantages and disadvantages. Historically, synthetic data has been associated with either the anonimization of data [1, 57] or the development and testing of software. In many cases, these overlapped. With the rise of big data and deep learning, private data has become a one of the most valuable types of data, and the promise of realistic synthetic data has so far been a holy grail. This is often because the utility and privacy of data are inversely proportional when it comes to machine learning models. The field of data synthesis with deep learning techniques is still in its infancy, and we expect to see promising results in the near future, as well as standardization of methods and evaluations.

Notation	Definition
p_z	Distribution of noise variable z
p_g	Distribution of generated data
p_r	Distribution of real data
x	Sample from p_r
z	Sample from p_z
G	The Generator
D	The Discriminator

Table 3.1: The following notations are used in this thesis.

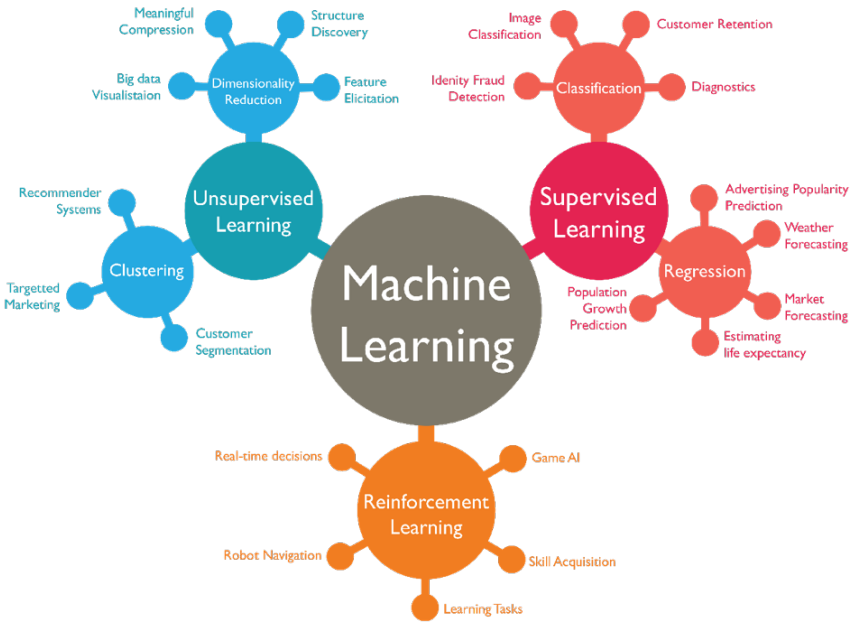


Figure 3.1: An overview of the different types of machine learning. On the right, there is supervised learning which uses labeled data to make predictions. On the left, we have unsupervised learning which uses unlabeled data and tries to find patterns and structures. At the bottom, we have reinforcement learning which tries to learn from its own experience.

3.2 Machine Learning

Machine learning is the capacity of machines to learn a specific task without explicit instructions, relying on patterns instead. Machine learning is considered a subset of the term Artificial Intelligence. There are three main types of machine learning: supervised learning, unsupervised learning and reinforcement learning.

3.2.1 Supervised Learning

Supervised learning [49] is done with labeled data, meaning that for each data point it is known to which class that data point belongs. This is used, for example, when performing classification and regression. Say we have a picture of a cat and a model that tells you whether the image contains a cat or a dog. The input data is the picture of the cat and since we know the picture contains a cat, the class (also called label) of the picture is 'cat'. Training is done by feeding the model the image and telling it how wrong the predicted class was. To achieve this ability, the data and classes need to be transformed to a format interpretable by a machine learning model. Often, this means the data and classes are transformed to numbers. For example, in this case we transform the class cat into a 0 while the class dog becomes a 1.

In learning tasks with tabular data, the data is often organised as follows. The rows contain data points while the columns contain features of those data points. When doing supervised learning, one column generally contains the class. Using the example of predicting one's salary using their personal information (age, ethnicity, number of children, etc), a row contains this personal information. When doing supervised learning, the class column indicates the salary of this person. This can be used as input to a machine learning model.

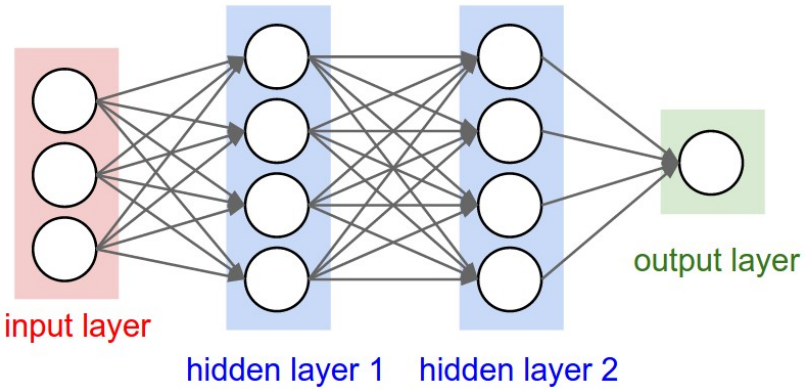


Figure 3.2: A example of a neural network, containing two hidden layers. The input layer contains the features that are used as input, while the output layer can represent several things, like a class probability or a scalar indicating some continuous value. From [19].

3.2.2 Unsupervised learning

The biggest difference between supervised learning and unsupervised learning [49] is the absence of classes. This means that this method is not suitable for classification or regression, but rather has applications in dimensionality reduction and clustering. Usually, dimensionality reduction tries to reduce the amount of data while preserving the same information, which is useful in, for example, image compression. Clustering can determine groups of data points that share characteristics, and are more closely related to each other than to the rest of the data points.

3.2.3 Reinforcement Learning

Reinforcement learning [56] can be considered a self learning model, where the model performs actions in a certain environment and receives feedback from the environment. To oversimplify it, this is similar to how humans learn. When a child get a reward for doing the right things but is being punished for doing the wrong things, it slowly learns. The same principle is applied in reinforcement learning. Reinforcement learning has seen a lot of usage in playing games with machine learning, like chess and go. In both cases, reinforcement learning was one of the fundamental blocks that allowed these models to perform so well.

3.3 Neural networks

Artificial Neural Networks (ANN), as is their full title, have existed since the 1940s [46, 25]. Neural networks are inspired by the workings of biological neurons from the brain. These neurons get a certain input and produce a certain output depending on the input. A neural network combines many layers of these neurons to create complex structures, which is able to learn advanced non-linear functions. The weights of the neuron's input and the threshold of the neuron are learned during training and are also called the parameters of the model.

After their gradual deaths in the 90s, also referred to as the *AI Winter*, neural networks were not used for quite some time due to the extremely long training times. But due to modern developments of hardware, most notably Graphics Processing Units (GPUs),

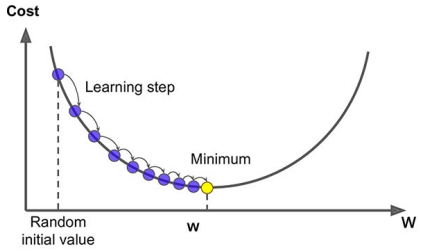


Figure 3.3: Example of the gradient descent algorithm. The parameter w is updated by calculating the gradient with respect to the loss ('cost' in the image) and multiplying it with a learning step. This is done until convergence, i.e. when w is at the minimum. From [9].

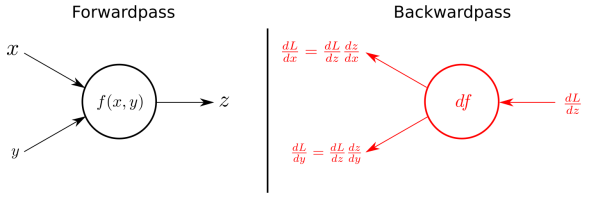


Figure 3.4: Example the forward and backward pass in a neural network. During the forward pass, we calculate the results by applying the function f on the input values x and y. In the backward pass, we calculate partial derivatives with respect to the loss with which we update the model parameters. From [38].

and some breakthroughs by LeCun et al. [42], Hinton and Nair [27] and Bengio, Lamblin, Popovici, and Larochelle [6], neural networks became popular again in the second part of the 00s. It would take another couple of years, but these hardware advancements combined with improved software, better understanding of neural networks, and huge amounts of data allowed for the rise of deep learning.

3.3.1 Backpropagation and Gradient Descent

One of the major building blocks of neural networks is their capacity to learn many parameters, sometimes millions. This capability is enabled by back propagation and gradient descent. A neural network is trained in two steps. First, some data is put into the network and produces a result. Second, it is measured how wrong the answer was (called loss or error) and the weights of the network are updated accordingly. The crux is in this update. Gradient descent enables a network to slowly converge to a minimum of the loss by iteratively calculating the gradients of the weights with respect to the loss and updating the network parameters. When the loss reaches its minimum, the network cannot improve with training anymore. The gradients are calculated with backpropagation, which is essentially the chain rule, applied from the output to the input of the network. By doing this many times of data, the network learns how to perform the task at hand. See figure 3.4 and figure 3.3.

3.4 Deep Learning

The term deep learning commonly refers to neural networks with several or many hidden layers. It originated in 2012, with the introduction of AlexNet [39]. AlexNet is a deep convolutional neural network which won the ImageNet challenge [14] (ImageNet Large Scale Visual Recognition Challenge in full, one of the standard computer vision datasets used for benchmarking) by a significant margin. Since then, neural networks have become one of the most researched subjects in artificial intelligence. Having the capacity to model essentially

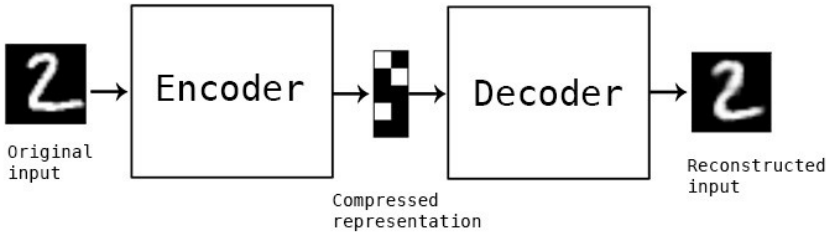


Figure 3.5: Overview of a standard autoencoder. The input is the image of an 2 from the MNIST dataset [41]. This digit is encoded to a lower dimensional latent space. The decoder then tries to reconstruct the input image from the encoding. The model learns how to do this by measuring the difference between the input and the output. From [3].

anything, neural networks have not only revolutionized computer vision but also natural language processing, speech processing and autonomous agents.

3.5 Deep Generative Models

The field of deep generative models is even younger than deep learning, with the inception of the *Variational Autoencoder* (VAE) [35] and *Generative Adversarial Network* (GAN) [20] in 2013 and 2014, respectively. Deep generative models are characterized by multi-layer neural network that are able to generate samples following the distribution of the data. The most important difference is that the VAE tries to model this distribution explicitly, which has the advantage of being able to do approximate posterior inference, i.e. given some data x, we can find the latent representation that caused this x. This will be explained more extensively in section 3.7. The GAN, on the other hand, tries to model the data implicitly which does not have this characteristic.

3.6 Autoencoder

Before discussing the variational autoencoder, some understanding of the vanilla autoencoder is required. An autoencoder consists of two parts: the encoder q and the decoder p. The encoder takes an input x and outputs a latent variable z of size n. The encoder has parameters θ. The encoder is formally represented as $q_\theta(z|x)$. Taking MNIST [41], a famous dataset consisting of 28x28 images of handwritten digits, as an example, the input of the encoder is an 28x28-dimensional image and this is encoded into a lower dimension of size n. When n is smaller than the dimension of x (which is generally the case), this corresponds to a compression of information into the lower dimension of z. The *decoder* p does the exact opposite and tries to reconstruct the original input x from the latent variable z. The network is often the exact opposite structure of the encoder network. The parameters of the decoder are denoted by ϕ. The decoder is formally represented as $p_\phi(x|z)$.

These networks are trained using the so called reconstruction loss, which often is the mean-squared error or cross entropy between the input and output. Intuitively it gives and indication how closely the output data resembles the input data.

The fundamental problem with normal autoencoders for generation is that the latent space of the encoded values may contain gaps, with the consequence that sampling this space

is not straight forward and interpolation might be meaningless. This is clear in Figure 3.6, where clusters are visible. It makes sense for the autoencoder to do, since these segments make it easier for the decoder to decode the latent vector z. However, it prevents from arbitrary sampling form the latent space and interpolating between points.

When dealing with a latent space with discontinuities, sampling the gaps essentially provides the decoder with data never seen before. The decoder has no idea what these latent vectors correspond to and will output unrealistic and unrecognizable data.

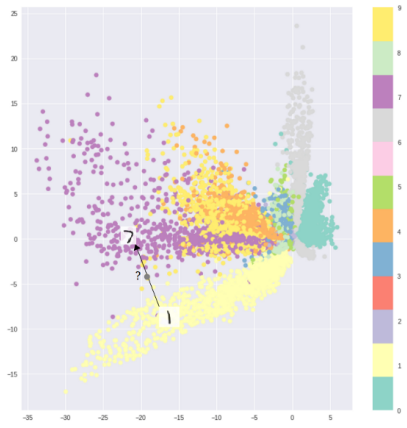


Figure 3.6: Latent space of autoencoder trained on MNIST, optimized solely for reconstruction loss [53]

3.7 Variational Autoencoders

The variational autoencoder was developed as an answer to modern data sampling requirements, like being able to handle complex models and large datasets. Where GANs have the capacity to generate very realistic samples, they have little control over which features you want present or omitted. A variational autoencoder on the other hand allows for very specific feature selection on generated data and also allows for altering of existing data.

The variational autoencoder fundamentally differs from the normal autoencoder because it does not have gaps in the latent space. This makes them excellent for generative tasks and allows for interpolation and transformations on real data. This is achieved by a rather surprising mechanism, namely splitting the latent vector of size n in two vectors of size n: one vector of means μ and one vector of standard deviations σ. Each value in the vector of means corresponds to a value in the vector of standard deviations at the same index. Combined, they define a probability distribution, generally a Gaussian, which is sampled to get input for the decoder. Say, we have a latent space of length 2; the encoder outputs two vectors, one with two means: $[0.3, 0.75]$ and one with two standard deviations: $[0.09, 0.04]$. We sample two Gaussians with those means and standard deviations to get the latent vector z: $[0.34, 0.74]$.

The result is that during training, the decoder not only sees the exact encodings of the input data, but sees samples from the distribution of the encoding. Intuitively, this means that the decoder learns that not only an exact points corresponds to a certain output, but the area surrounding it as well which contains very similar data points.

Because there are no limits to μ and σ, they could end up as many small 'islands' in the latent space, keeping the problem of the gaps. Ideally, we want these distributions to be as close to each other as possible while still being separate to allow for smooth interpolation, arbitrary sampling. To achieve this, the Kullback-Leibler Divergence (KL-divergence) is introduced as part of the loss function. The KL-divergence is a measure of how *diverged* two probability distributions are. Minimizing this means to optimize the parameters of the probability distribution μ and σ to resemble the real distribution as closely as possible. The

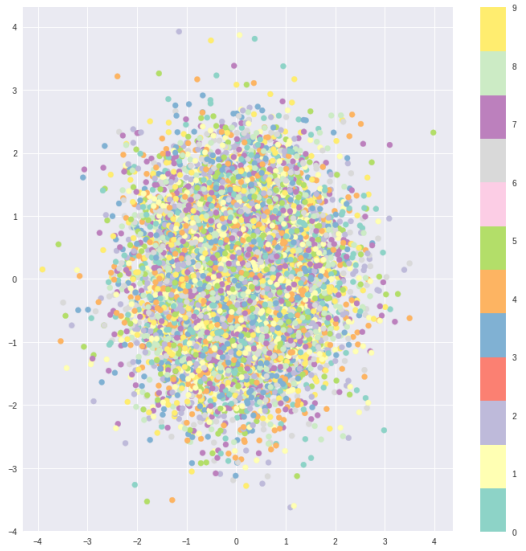


Figure 3.7: Latent space of an autoencoder trained on MNIST, solely optimized for KL-divergence loss [53]

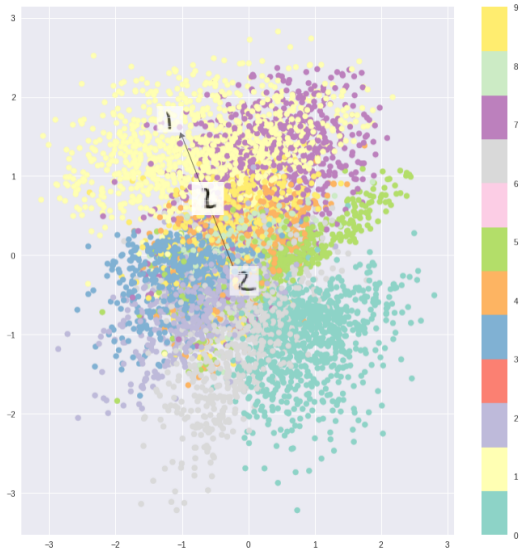


Figure 3.8: Latent space of an autoencoder trained on MNIST, optimized on the combined reconstruction loss and KL-divergence [53]

KL-divergence is defined as

$$KL(p \parallel q) = \sum_{i}^{N} p_i \cdot log\left(\frac{p_i}{q_i}\right)$$

where p and q represent two distributions.

Because the loss function is a sum of errors from individual data points we can decompose it to just one data point. The total loss function is

$$L = \mathbb{E}[\log p(x|z)] - KL(q(z|x) \parallel p_z)$$

In this loss function the first part is the reconstruction error between the input and output and the second part is the just mentioned KL-divergence. Intuitively, the KL-divergence rewards the encoder for putting values around the center of the latent space, distributed according to the density function. However, it has little regard for clustering similar images and results in distributions as seen in Figure 3.7

The magic happens when we combine both losses. The results is a densely packed latent space which maintains the similarity among local encodings and maintains the global segmentations. This is shown in Figure 3.8.

3.8 Generative Adversarial Networks

GANs [20] were introduced by Ian Goodfellow et al. in 2014. Their paper introduced a novel neural architecture based on game theory where two adversarial "players" compete in a minimax game. Their proposed architecture sidesteps some of the problems of explicitly modeling a data distribution, like approximating intractable probabilistic calculations that

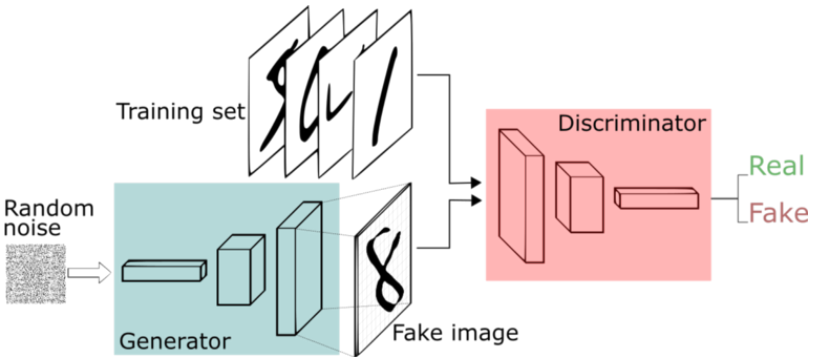


Figure 3.9: Overview of the GAN architecture with MNIST images as examples. The generator gets some random noise as input and learns to map this to the distribution of the real data. The output of the generator goes into the discriminator, along with a real image. The discriminator then tries to tell which image is fake and which is real. From [10].

occur in maximum likelihood estimation with complex functions (Such as in the VAE). The GAN framework consists of a generative model, pitted against an adversary: an discriminative model that learns to determine whether the samples originated from the generative model or the real data. This competition drives both parties to improve their methods until the fake data points are indistinguishable from the real data points.

In this and the coming subsections, we outline the vanilla GAN architecture, the WGAN architecture and the WGAN-GP architecture. The GAN and WGAN-GP architecture are used in my experiments and the WGAN is required background knowledge for understanding the WGAN-GP. These architectures are all suited for "normal" generation, meaning they were proposed to simply generate samples from the implicit distribution learned from the real data. There are other models who allow for conditional training and sampling like CGAN[48] and InfoGAN [11]. In the case of CGAN, this is achieved by simply appending a one-hot encoded vector to the input of both the generator and the discriminator. For example, you can let a generator trained on images of cats and dogs generate only images of cats. The capacity to conditionally model data often corresponds to having some control over the presence of certain features. InfoGAN specifically targets disentangling features, allowing for fine control of features like subject, rotation and shift in images. Furthermore, there are style transfer architectures that enable the copying and pasting of a specific style on a target image or between two images like CycleGAN[61] and Pix2Pix GAN[31], enabling very impressive results like real time video style transfer. Formally, a GAN consist of a generator G that tries to learn the distribution over data x using an random noise variable z. To learn, a prior is defined on the random noise variable p_z. This p_z can be any distribution and in the original GAN was represented by a uniform distribution. However, in more recent research a Gaussian distribution is more common. A mapping function is defined from the prior z to the data space with $G(z;\theta_g)$, where G is a differentiable functioned represented by a neural network with parameters θ_g. Additionally, a second neural network $D(x;\sigma_d)$ is defined, which maps from its input data x to a single value which represents the probability that x came from the input data, rather than $G(z)$, which is called the discriminator. The parameters of the discriminator are indicated with σ_d For ease of use, from now on p_z refers to the noise prior, p_g refers to the data distribution created by the generator and p_r to the distribution of real data. The minimax game that is played by the two is defined by the

Figure 3.10: Translation of an image with the Pix2Pix GAN [31]. The Pix2Pix GAN gets an input image from some domain and learns to map that to a different domain. In this example, the drawing of the handbag is mapped to a realistic image of a handbag.

following value function

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_r}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))].$$

which is essentially the same as minimizing the Jensen Shannon Divergence (JS-divergence). The JS-divergence is build up from the already familiar KL-divergence and is defined as follows

$$JSD(p_r,\ p_g) = \frac{1}{2}KL(p_r \parallel m) + \frac{1}{2}KL(p_g \parallel m)$$
$$\text{where } m = \frac{1}{2}(p_r + p_g)$$

The above formula results in the two following loss functions for the discriminator (L_d) and generator (L_g), respectively.

$$L_d = \log D\left(x\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}\right)\right)\right)$$
$$L_g = \log\left(1 - D\left(G\left(z\right)\right)\right)$$

A drawback of this specific implementation is that the JS-divergence does not always provide usable gradients. Most notably, when the discriminator D comes close to the optimal discriminator $D^\star$, the JS-divergence saturates and the gradient for the generator vanish. To circumvent this problem, Goodfellow et al. proposed a slightly adapted loss function for the generator as follows

$$L_g = -log(D(G(z))$$

Even though this adaptation fixes the previously mentioned problem, it introduces new ones of its own which we will go into in a minute. The adapted loss function is based on a reverse KL-divergence and a JS-divergence term as follows

$$L_g = KL(p_g \parallel p_r) - 2 \cdot JSD(p_g \parallel p_r)$$

Where, again, P is the data and Q are generated samples. Note that the terms are different than the earlier KL terms in the JSD formula. This KL term has the generated data as first parameter and the real data as second parameter. This adapted loss function penalizes unnatural new data by means of the KL-divergence term, but does not enforce diversity which may lead to mode collapse.

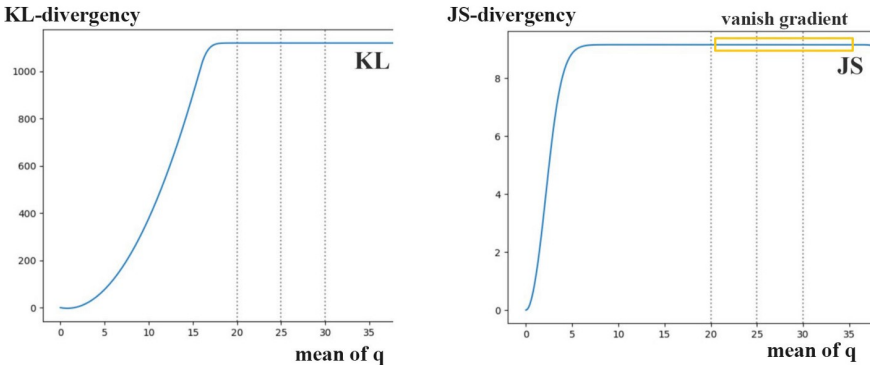


Figure 3.11: Vanishing gradients when using KL and JS divergence loss with increasing distance between p_r and p_g. It becomes clear that when the distance between p_r and p_g is large, the gradient vanishes. From [29]

3.8.1 Mode Collapse

Mode collapse is a phenomenon seen in GANs where the generator learns to output only a subset of the real data. For example with MNIST, a generative model might only output the numbers 4 and 7. These outputs can be quite realistic, but globally do not approach the distribution of the real data. Mode collapse in one of the major problems still to be solved with GANs. Complete mode collapse is not very common, but partial mode collapse is. Although the debate as to how mode collapse exactly works is still ongoing, we'll illustrate what the common suspicion is.

The objective of the generator is to provide samples that fool the discriminator the most. Looking at the loss function $L_g = log(1 - D(G(z))$ it is clear that the generator gets a low (good) loss when the discriminator outputs a value close to 1 for the generated data. Consider a situation where we're training a GAN for MNIST images. If we train G without updating D, G will converge to the optimal image $\boldsymbol{x}^\star$ where it generates only images that fool D. From the perspective of the discriminator, this $\boldsymbol{x}^\star$ is the most realistic image. In this extreme case, $\boldsymbol{x}^\star$ becomes independent of z. This means, no matter what z goes into G, G always outputs $\boldsymbol{x}^\star$.

$$\boldsymbol{x}^\star = argmax_x D(x)$$

The mode collapses to a single point: total mode collapse. As a consequence, the gradient of z approaches zero and the GAN is stuck. When restarting training of the discriminator, the most effective way for the discriminator to detect images from G is to detect the single point $\boldsymbol{x}^\star$. Because the generator has desensitized itself from z, the gradient from D will likely push the single point to the next most vulnerable mode. Due to the imbalance in the samples generated by G, it loses the capability to detect other modes. Both networks are overfitting to the weakness of the opponent. This results in a cat-and-mouse game between modes and the model will not converge. This phenomenon is illustrated in Figure 3.12

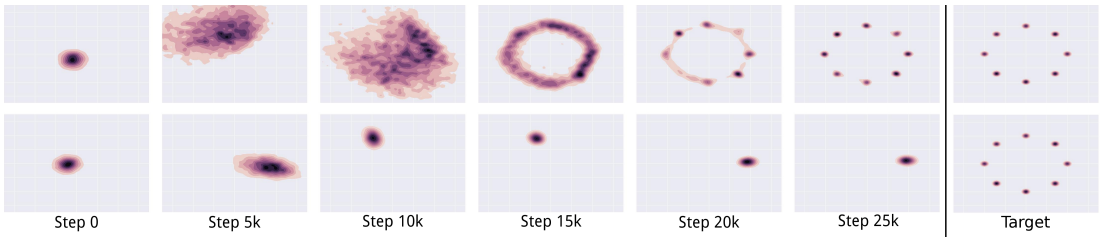


Figure 3.12: Mode collapse illustrated on a 2D toy dataset. We see the collapsing GAN trying to find a point that can be exploited, but the discriminator keeps learning that this weakness is being abused, and corrects itself. This pattern repeats itself while never converging. From [47].

3.9 Wasserstein GAN

The Wasserstein GAN (WGAN in short) is an adaptation from the vanilla GAN, proposed by Arjovsky et al. [5] in 2017. They identified many of the problems mentioned in section 3.8 with the original architecture and try to improve it by having a meaningful distance measure as loss function, namely the 1-Wasserstein distance. This seemingly small change results in a model that not only trains more stable, but also suffers less from mode collapse and the end result is often also better. So lets dissect why this is.

3.9.1 1-Wasserstein Metric

The 1-Wasserstein metric is a measure for the distance between two probability distributions in a given metric space. The 1-Wasserstein metric is also called the Earth Mover's distance (EM), because it is best explained intuitively as having two piles of dirt, where the cost of turning one pile into the other is corresponds to the amount that needs to be moved times the distance it needs to travel.

From here on, we will refer to the 1-Wasserstein metric as either just the Wasserstein metric or the Earth Movers distance. The 1-Wasserstein metric is defined as follows:

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\, \|x - y\| \,\big] \tag{3.1}$$

where, like before p_r and p_g denote the distributions of the real and fake data, respectively. Furthermore, $\prod(p_r, p_g)$ denotes all possible joint distributions $\gamma(x, y)$ whose marginals are p_r and p_g, respectively. Intuitively, $\gamma(x, y)$ denotes the amount of mass that needs to be transported from x to y to transform the distribution p_r into p_g. The EM distance is then the cost that is associated with the optimal transport plan.

The EM distance has two main advantages over alternatives like the JS-divergence. In their paper, Arjovsky et al. [5] prove, under certain restrictions, the following: given a function $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ where $\mathcal{Z}$ denotes the space of the latent input z and $\mathcal{X}$ denotes the space of the real data.

1. If g is continuous, then so is $W(p_r, p_z)$
2. If g is locally 1-Lipschitz[1], then $W(p_r, p_z)$ is continuous everywhere and differentiable almost everywhere.

[1] A differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere.

Both statements are false for the JS-divergence, making the EM distance a much more applicable measure as a loss function.

In figure 3.13, it is clearly visible that the Wasserstein metric provides smoother gradients, even when the generator provides very poor samples. This is in stark contrast to the gradients provided by the vanilla GAN discriminator, which provides no gradients if the real and fake data distributions are far apart.

3.9.2 Limitations

However, the equation for the Wasserstein distance is intractable, because to compute the infimum $\inf_{\gamma \in \Pi(p_r, p_g)}$ from equation 3.1, one needs to exhaust all the possible joint distributions. Using the Kantorovich-Rubeinstein duality, the calculation can be simplified to

$$W(p_r, p_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

where sup is the least upper bound and f is a 1-Lipschitz function with the constraint $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$. The only step left until we can calculate the Wasserstein distance is to find a 1-Lipschitz function. Creating a function for the task is something a neural network is very well suited for, but it does not adhere to the 1-Lipschitz constraint by itself. This is why the authors clip the weights during training:

$$w \leftarrow clip(w, -c, c)$$

As is also noted by the authors, this is obviously a sub optimal measure to comply with the 1-Lipschitz constraint. When clipping, the model will take longer to converge on the and the final performance is related with choosing a good c. The model is quite susceptible to changes of c, where a good c will provide a good network, but a bad c provides a network that does not converge and produces poor results. The clipping measure was chosen for its simplicity, but is something that can easily be improved upon.

3.9.3 Usage in WGAN

In the WGAN, the discriminator does not have a sigmoid activation at the end layer and the Wasserstein metric is applied to the output of the last layer of the discriminator. In deep learning, outputs without a activation are often referred to as the logits and applying the Wasserstein metric on this forces the logit distributions to be similar. The score correlates to how real the discriminator thinks the samples from G are, but is not a probability. The literature renames the discriminator to a *critic* to reflect this change, but we will keep using the shorthand D in formulas for both the WGAN and WGAN-GP.

Making it concrete, the loss functions of the WGAN are

$$L_D = \frac{1}{m} \sum_{i=1}^{m} D(x) - \frac{1}{m} \sum_{i=1}^{m} D(G(z))$$

$$L_G = \frac{1}{m} \sum_{i=1}^{m} D(G(z))$$

which are both easily calculable values. Additionally, mode collapse was not observed in their experiments and does not seem to occur when using the Wasserstein distance.

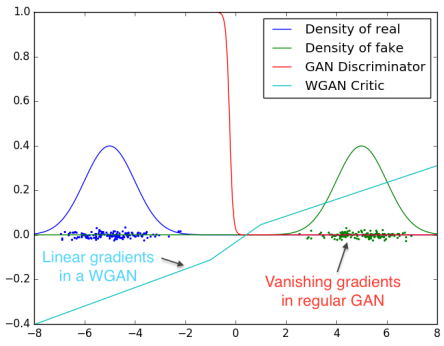


Figure 3.13: Wasserstein GAN loss versus vanilla GAN loss. It becomes clear that the Wasserstein distance as an error function provides much more stable gradients when the data distributions of the real and fake data are far apart. From [5].

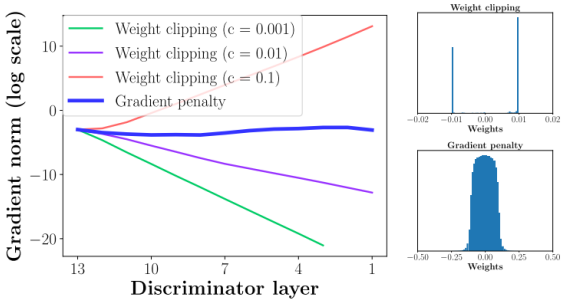


Figure 3.14: Gradients of the critic for different values of c. The gradient penalty will be discussed in section 3.10, but in short; it is an improvement on the clipping measure of the WGAN which greatly improves the capacity to learn. Instead of most weights being either the minimum or the maximum, we get a smooth distribution. From [23].

As a last measure, it is observed that the usage of momentum in the optimizer results in unstable gradients when using the Wasserstein distance. They solved this by setting β_1 in Adam [34] to 0, making it the same as RMSProp [21]. In summary, the WGAN has two major advantages over the vanilla GAN architecture:

1. it shows no sign of mode collapse
2. the generator can still learn when the critic performs well and p_r and p_g are far apart.

3.10 WGAN-GP

WGAN-GP is short for Wasserstein GAN with Gradient Penalty, and was introduced by Gulrajani et al. [23] shortly after the original WGAN paper. It proposes a gradient penalty to comply with the 1-Lipschitz constraint, removing the need for clipping. WGAN-GP increases and decreases the gradient penalty depending on how far the gradient is from being 1-Lipschitz. They change the loss function of the WGAN to

$$L_d = \underbrace{\mathbb{E}\left[D(x)\right] - \mathbb{E}[D(G(z))]}_{\text{Original critic loss}} + \underbrace{\lambda \cdot \mathbb{E}_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}}(D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient penalty}}$$
$$\text{with } \hat{x} = \epsilon x + (1-\epsilon)G(z)$$
$$\text{and } \epsilon \sim U[0,1]$$

where λ is set to 10. This λ was empirically found to be effective across a variety of architectures and datasets. This change requires some additional changes to the architecture, most notably the omission of batch normalization in the critic. In other work, batch normalization is generally encouraged because it stabilizes training. However, batch normalization changes the form of the discriminator problem from mapping a single input to a single

ARCHITECTURE	MNIST	FASHION	CIFAR	CELEBA
MM GAN[20]	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN[20]	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN[45]	7.8 ± 0.6	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8
WGAN[5]	$\mathbf{6.7 \pm 0.4}$	$\mathbf{21.5 \pm 1.6}$	$\mathbf{55.2 \pm 2.3}$	$\mathbf{41.3 \pm 2.0}$
WGAN GP[23]	20.3 ± 5.0	$\mathbf{24.5 \pm 2.1}$	$\mathbf{55.8 \pm 0.9}$	$\mathbf{30.0 \pm 1.0}$
DRAGAN[37]	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN[8]	13.1 ± 1.0	$\mathbf{22.9 \pm 0.9}$	71.4 ± 1.6	38.9 ± 0.9
VAE[35]	23.8 ± 0.6	58.7 ± 1.2	155.7 ± 11.6	85.7 ± 3.8

Table 3.2: FID scores for different GAN architectures on four well known datasets. It shows the WGAN and WGAN-GP beating out most of the other architectures. From [44].

output to mapping a batch of inputs to a batch of outputs. Since the gradient penalty is calculated with respect to each individual data point and not the entire batch, the gradient penalty would not be valid anymore. Experiments show that the model still trains well without batch normalization. Normalization techniques that do not introduce correlations between samples are still available, like layer normalization, which is used as a drop-in replacement for batch normalization.

To show that this method actually works; in a study from Google Brain, they compared models with different training methods and loss functions. They trained the listed models to synthesize images from the four datasets and evaluated it using the Fréchet Inception distances (FID). FID is a metric to determine how realistic an image is, where lower scores correspond to more realistic images. WGAN and WGAN-GP achieve some of the best FIDs, especially on larger datasets (CIFAR, CELEBA). Table 3.2 shows how different architectures stack up against each other.

3.11 GANs for Tabular Data

So far we have only discussed GANs in general, which are commonly applied to images. However, this thesis goes into GANs applied to tabular data so that will be discussed extensively in this section. GANs for tabular data have numerous extra hurdles to clear for generating data, like transforming data to be applicable for GAN usage, being able to handle different data types simultaneously and doing an deterministic inverse transformation of generated data. So compared to discriminative models and GANs for image data, this requires a couple more steps. In this section we will walk through the steps that are required to train GANs on tabular data.

As discussed in the introduction, in this project we focus on the two most common data types: categorical and continuous. No distinction between nominal and ordinal data is made, so, for example, gender will have the same encoding as education, although education clearly has an order with higher and lower levels.

3.11.1 Preprocessing

Preprocessing is limited when working with GANs. Rows with empty values are removed, because we have enough data. Additionally, we make a judgement of which columns we want to generate with the GAN. Some columns will make sense to generate, while others will not. For example, generating unique IDs with a GAN makes no sense, since their is no correlation with any of the attributes and GANs are not suited for generated unique data. These columns are better removed and than added after the process again with an appropriate method. When generating unique IDs, generating k numbers could be done by generating all numbers from 1 to n where $n > k$ and then sampling k numbers without replacement from this set. If the numbers need to be in a certain range, say between i and j, the same principle can be applied starting from i instead of from 1.

3.11.2 Data transformations

To generate meaningful data, the data needs to have, like in all neural networks, an appropriate representation. There are different ways to approach this for the two data types. Normalizing continuous values can be done in many ways, but is often done by using standardization or normalization. Standardization transforms the data to have mean 0 and standard deviation 1 while normalization transforms all data to a value between 0 and 1. Standardization is generally done with $x'_r = \frac{x_r - \mu}{\sigma}$ where x_r is the real data, x'_r the transformed data, μ is the mean and σ is the standard deviation. One can multiply σ with some scalar to capture more or less of the distribution within $[-1, 1]$, with 4σ capturing 99.99% of all data, if the data follows a Gaussian distribution. However, there are many alternatives like min-max-scaling, binning into discrete values, using outlier reduction, and Gaussian mixture models.

Categorical values are a bit of a different story, because in discriminative models these are typically represented with learned embeddings. This approach has proven to be extremely powerful and is the core that is powering the natural language processing renaissance [15, 52]. However, inverse transformations from generated embeddings require a distance measure to see which real embedding is closest, and distance measures are often ambiguous when applied in high dimensional spaces.

Detecting which columns corresponds to which data type is not trivial, which is why many implementations require this to be given alongside the dataset. The effect of this approach is that results are easily reproducible while reducing flexibility. To use a new dataset, one needs to specify the data types for the all the columns, if categorical, define how many values there are and what the labels corresponding with the numerical encoding are.

3.11.3 Reverse transformation

Because the output of a GAN is within a very specific domain, typically between -1 and 1, the output needs to undergo an inverse transformation with respect to the initial transformation. This ensures that the output data looks like the original data. For example, the output of the GAN might be a one-hot encoded vector which needs to be transformed to represent one of several classes. Additionally, a value of 100 is commonly scaled to a value in the range $[-1, 1]$ before being used in a neural net. After generation, an inverse transformation is required. The inverse transformation is tailored to the initial transformation, being the exact opposite.

3.11.4 GANs for categorical data

Generation of categorical data in GANs has been an issue with several proposed solutions and workarounds. The problem arises due the the enforcement of a hard decision at the end of the generator for categorical values. The output of a generator with categorical values is often given with a softmax layer, indicating which output is most likely to be correct. However, the data is one-hot encoded and if we were to give the softmax outputs and the one-hot encoded data to the discriminator, it could easily tell the difference.

There are two workarounds for this problem.

1. The first one is to have an $onehot(argmax(x))$ applied to the output of the network. For example, $[0.01, 0.06, 0.12, 0.81]$ becomes $[0, 0, 0, 1]$. The problem with this approach is that $argmax$ does not have a derivative, which is a requirement for back propagation to work. To work around $argmax$, one can use the Gumbel softmax [33], which is an adaptation of softmax that pushes the softmax values to the limits 0 and 1. Like in the variational autoencoder, the Gumbel softmax uses a reparameterization trick to create a differentiable approximation of $argmax$, leveraging softmax with temperature. Gumbel softmax is calculated with

$$y_i = \frac{\exp((\log \alpha_i + G_i)/\tau)}{\sum_{j=1}^{K} \exp((\log \alpha_i + G_j)/\tau)}$$

   where α is the input, y the output, G_i represents a sample from the Gumbel distribution and τ is the temperature parameter, which controls how much the values are pushed apart or together. Using $\tau > 1$, the softmax values come closer together and provide smoother gradients, which is desired at the beginning of training. Annealing τ during training pushes the values closer to a one-hot encoded value, which is desired at the end of training.

2. The second one is to apply noise to the onehot-encoded ground truth, so they look more like the softmax output. The underlying assumption is that a discriminator would not be able to easily distinguish between the noisy ground truth and the output of the generator.

Both approaches are used throughout the literature, but recently the trend has been going towards using the Gumbel softmax.

Chapter 4

Related Work

In this section we discuss three main topics: generation methods, evaluation metrics and baselines. Starting with the generation methods, we will look at the current state of the art contenders and what approaches they take. We detail the four highest performing models extensively.

4.1 Generation Methods

4.1.1 TGAN

TGAN is a architecture for tabular data generation proposed by Xu and Veeramachaneni [58]. The authors' goal was to provide a package capable of generating all datasets with categorical and continuous data. They do this with a generator that has an RNN with LSTM-cells that walks over the columns and predicts the value for the next column depending on earlier outputs. The output of the LSTM is put through a couple of dense layers, including an attention layer to get the final output. The discriminator is much simpler, just being a fully connected network. Furthermore, they identify the problems neural networks have with non-Gaussian distributed inputs and propose a probabilistic distribution method to circumvent these. More specifically, they use a Gaussian Mixture model (GMM) applied on each numerical column individually. They call this approach mode-specific normalization for numerical variables.

The GMM is trained with m components for each numerical column C_i. The result of fitting the GMM are the the means and standard deviations of the m components, $\mu_i^{(1)}, \ldots, \mu_i^{(m)}$ and $\sigma_i^{(1)}, \ldots, \sigma_i^{(m)}$ respectively. For each value in the column $C_{i,j}$, they compute the probability coming from each of the m Gaussian distributions as a vector $u_{i,j}^{(1)}, \ldots, u_{i,j}^{(m)}$. So the final vector $u_{i,j}$ is a normalized probability distribution over m Gaussians. As a last step, the values are normalized by the mean and standard deviations of the Gaussian to which they most likely belong: $v_{i,j} = (c_{i,j} - \mu_i^{(k)})/2\sigma_i^{(k)}$. As a very last step, these values are clipped to $[-0.99, 0.99]$ before they are used as input for the neural network. In the end, each value $c_{i,j}$ is represented by the combination of $u_{i,j}$ and $v_{i,j}$

To make this a bit more concrete, please have a look at figures 4.1 and 4.2 to get some feel for this encoding. Each data point is encoded by their probability of belonging to each of the four Gaussians, as well as their relative position within the most probable Gaussian.

Regarding the problems with categorical data, their implementation does not suffer from the problem with the derivative of $argmax$ being 0 because they do not apply $argmax$.

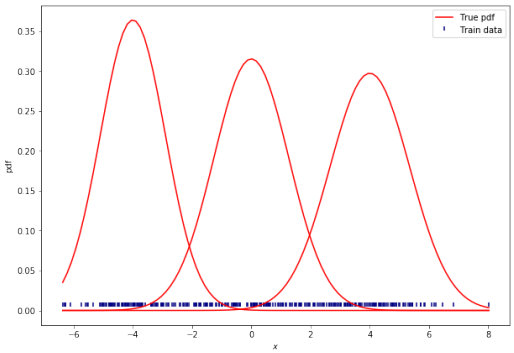


Figure 4.1: A set of data points created by sampling three Gaussians 100 times each. The results is a data distribution that is very hard to model with a single Gaussian. From [4].

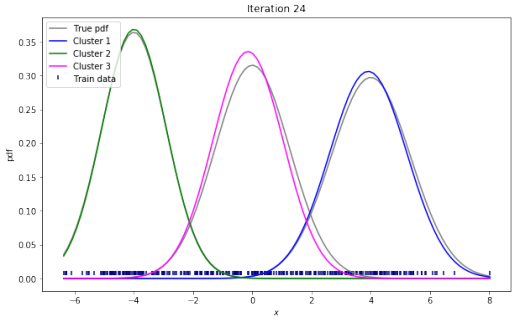


Figure 4.2: Three Gaussians fitted to the sampled data gives us three almost identical Gaussians. If this were modeled using a single Gaussian, sampled data would be very different than the real data. From [4].

Instead, they keep the normal softmax output. To close the gap that clearly exists between a softmax distribution and a one-hot encoded real value they apply label smoothing on the real data, which makes it harder for the discriminator to distinguish the two. In practise, this seems to work quite well and they achieve fairly impressive results.

Additionally, they propose a regularization measure with what they call a diversity vector. Each dimension in this vector consists of the total distance between one sample and all other samples in the mini-batch with a predefined metric. This gives the discriminator some meta information about that specific sample. However, they do not specifically evaluate this addition, so it is hard to say what and how much effect this has.

Lastly, they use the normal GAN losses, but add an KL-divergence term to the generator loss to prevent overfitting. As discussed in section 3.8, this likely has a desired effect on the sample diversity but hurts sample quality. The loss function of the TGAN generator is

$$L_G = -\mathbb{E}_{z\sim U(0,1)} \log(D(G(z)) + \sum_{i=1}^{n_c} KL(u'_i, u_i) + \sum_{i=1}^{n_d} KL(\mathbf{d}'_i, \mathbf{d}_i)$$

where n_c and n_d indicate the number of continuous and categorical columns, respectively, while, $\mathbf{d}$ indicates the one-hot vector of categorical columns.

In this thesis, we will propose two adaptations of the TGAN model that improve the stability of training and the quality of generated samples.

4.1.2 MedGAN

MedGAN[12] takes a wildly different approach. They propose using an autoencoder to translate categorical values to a latent continuous representation which can be learned by the generator. Using a pretrained autoencoder (which consists of *Dec* and *Enc*), the decoder sits between the generator and the discriminator and translates the continuous output of the generator to the format of the real data. Their implementation had a couple of limitations. For example, they only implemented and evaluated for binary and what they call count variables. Count variables are essentially ordinal integer values, but do not seem to have a predefined limit. Additionally, their implementation does not support handling of

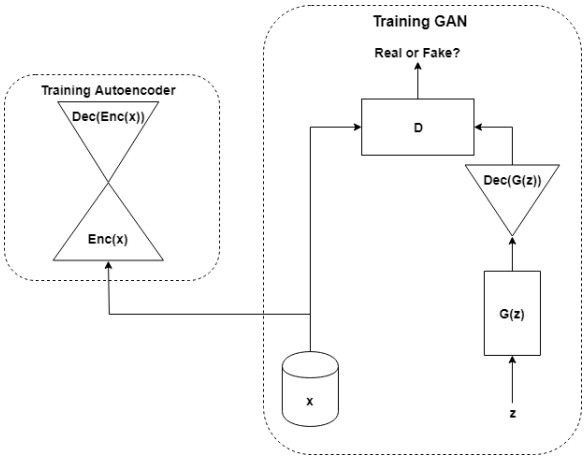


Figure 4.3: The architecture of MedGAN. First, the autoencoder is trained, which is visible left. Then, the GAN is trained with the decoder of the autoencoder situated between the generator and discriminator.

both data types at the same time, but rather in separate models.

Before getting into the gritty details, lets talk about their contributions.

1. They propose a method to circumvent categorical values in GANs by using autoencoders.

2. They propose a technique called minibatch averaging, where they use the average of a minibatch as additional input for the discriminator. This helps in reducing mode-collapse and serves a similar purpose as the diversity vector used in TGAN.

3. They add skip connections to the dense layers in their generator G. These are the same kind of connection as in ResNet and other state-of-the-art discriminative models and allow for the model to look back at earlier information as well as improving the flow of gradients.

In their results, they present results both with regard to classifier F1 scores and to privacy capabilities. For the two data types mentioned, MedGAN seems to outperform the evaluated alternatives like VAE, and Restricted Boltzmann Machines (RBM) [55]. Additionally they evaluate on several artificial baselines like sampling from a Bernoulli distribution, based on presence in the real dataset and randomly flipping a binary value in 10% of the cases. However, they do seem to improve on the VAE and RBM, which is meaningful on its own. Additionally, they perform a quite extensive evaluation with regards to the protection of privacy.

All the parts of MedGAN consist of multilayer perceptrons with different activations, depending on the datatype. For count variables, they use ReLU in both the *Enc* and the *Dec*. For binary values, they use tanh in the *Enc* and sigmoid activations in the *Dec*.

Surprisingly, contradicting what was discussed in section 3.6 about sampling from a latent space with gaps, this approach seems to work well for MedGAN; we return to this topic in chapter 7. We chose not to built upon MedGAN for two reasons: the model was only functional with specific data types, while we required more, and the model reported performance that was much lower than TGAN and TableGAN.

4.1.3 TableGAN

TableGAN[50] takes a very different approach, and tries to capture correlations in 2D using convolutional neural networks. We will walk through their approach step by step.

In TableGAN, data encodings are much more straight forward than in TGAN. categorical values are transformed to numerical values which is than minimax scaled to a value in $[-1, 1]$ (i.e. $[a, b, c] \rightarrow [1, 2, 3] \rightarrow [-1, 0, 1]$) for usage with tanh in the final layer. Continuous columns are just minimax scaled to between -1 and 1.

The TableGAN architecture is based on that of the Deep Convolutional GAN (DCGAN), which was one of the key drivers for high fidelity image synthesis. In the generator G, all intermediate layers use ReLU, as is common in convolutional models while the discriminator D uses LeakyReLU for all except the last layer, which uses sigmoid.

They introduce a classifier C, identical to discriminator D in its design. However, C does not classify whether the sample is real or fake, but uses one of the columns as labels which it tries to predict. In their testing, this seems to help sample quality and coherence. They give the example that someone with a cholesterol of 50 cannot have a label of diabetes, because cholesterol of 50 is too low to be diagnosed with diabetes.

The loss of this classifier C is combined with the normal GAN loss and an additional loss, called *information loss*. The information loss is closely related to the Wasserstein distance, and is evaluated over the output of the last layer before the sigmoid in the discriminator. Where the Wasserstein distance is defined as the absolute difference in means of the logits, the information loss is defined as follows:

$$
\begin{aligned}
L_{mean} &= \|\mathbb{E}[\mathbf{f}_x]_{x \sim p_r} - \mathbb{E}[\mathbf{f}_{G(z)}]_{z \sim p_z}\|_2 \\
L_{\sigma} &= \|\sigma[\mathbf{f}_x]_{x \sim p_r} - \sigma[\mathbf{f}_{G(z)}]_{z \sim p_z}\|_2 \\
L^G_{info} &= L_{mean} + L_{\sigma}
\end{aligned}
$$

where $\mathbf{f}$ represents the logits in the last layer of the discriminator and σ represents the standard deviation. So if $l_{mean} = 0$ and $l_{\sigma} = 0$, the real and synthetic data look identical to the discriminator, which will not be able to distinguish them from each other.

The authors identify a need to control the privacy of the real data points, where a distribution of synthetic data identical to the real data is the least private and one that deviates from the real data distribution more private. They control this by adapting L^G_{info} to

$$
L^G_{info} = max(0, L_{mean} - \delta_{mean}) + max(0, L_{\sigma} - \delta_{\sigma})
$$

where δ_{mean} and δ_{σ} are two hyper parameters to control the privacy. This regularization sets $L^G_{info} = 0$ as long as L is smaller than δ for both the mean and standard deviation losses. If δ_{mean} and δ_{σ} are 0, the loss in unconstrained and the synthetic data distribution will try to match the real data distribution. This adaptation seems to accomplish exactly what it was set out to do, with a strong relation between sample quality and δ_{mean} and δ_{σ}.

In our testing, TableGAN works quite well for continuous values, but suffers in situations with categorical values, which is why we chose TGAN over TableGAN.

4.1.4 CTGAN

Conditional Tabular GAN [59] is a follow-up paper of [58], but does not directly continue where that paper left of. This paper introduces another tabular GAN architecture, along with new data preprocessing steps that improve tabular GANs. Confusingly, they have called this architecture TGAN as well, but have since renamed it to CTGAN in the accompanied python package so this is how we will refer to it as well.

The paper starts strong, by identifying five crucial problems that need to be addressed for successful tabular data synthesis.

1. **Mixed data types.** Real data often consists of mixed data types. To generate this data simultaneously, a GAN must be able to apply both softmax and tanh on the output.

2. **Non-Gaussian distributions.** In images, values generally follow a Gaussian-like distribution, which can easily be normalized to $[-1, 1]$, whereas with tabular data, continuous data often have non-Gaussian distributions with long tails. With tanh and sigmoid, the location of the gradients will often be flat and a phenomenon called gradient saturation occurs. When gradient saturation occurs, a model loses the ability to learn through gradients.

3. **Multimodal distributions.** Tabular continuous data typically have multiple modes. They find that 57/123 continuous columns in their test datasets have multiple modes. This and the previous problem can be overcome with some specific preprocessing steps which will be detailed in a minute.

4. **Learning from sparse one-hot encoded vectors.** The problem of softmax appears again, which is now solved by using the mentioned Gumbel-softmax [32].

5. **Highly imbalanced categorical columns.** Many categorical columns have a highly imbalanced distribution, meaning the major category appears on more than 90% of the rows. Missing the minor classes does not impact the overall distribution much, and is often not noticed by the discriminator. In essence, this is a form of mode-collapse, for which they apply PacGAN [43], in which multiple samples are presented together to the discriminator to determine whether they are fake or real in total.

The data preprocessing steps are quite alike the ones of TGAN [58], with some small adaptations. They still encode all continuous values, but instead of the normal Gaussian mixture model, they use the variational Gaussian mixture model (VGMM). The VGMM model has additional parameters $u^{(1)}, \ldots, u^{(m_i)}$, which are weights. Please recall that C_i denotes column i and $c_{i,j}$ denotes a value in column C_i.

1. At first, we try to estimate the number of modes in the distribution of C_i. We use the VGMM to do this. This produces the probabilistic model $\mathbb{P}_{C_i}(C_{i,j})$ and a Gaussian Mixture with m_i components with means $\mu_i^{(1)}, \ldots, \mu_i^{(m_i)}$, standard deviations $\sigma_i^{(1)}, \ldots, \sigma_i^{(m_i)}$ and weights $u^{(1)}, \ldots u^{(m_i)}$.

$$\mathbb{P}_{C_i}(c_{i,j}) = \sum_{k=1}^{m_i} u^{(k)} \mathcal{N}\left(c_{i,j}; \mu_i^{(k)}, \phi_i^{(k)}\right)$$

   The VGMM is trained to maximize the likelihood on the training data.

2. Compute the probability mass function (PMF) of the value $c_{i,j}$ sampled from each of the m_i modes as

$$Cat(k; [\bar{\beta}_{i,j}^{(k)}]_{k=1\ldots m_i}), \text{ where } \bar{\beta}_{i,j}^{(k)} = \frac{u^{(k)} \mathcal{N}\left(c_{i,j}; \mu_i^{(k)}, \phi_i^{(k)}\right)}{\mathbb{P}_{C_i}(c_{i,j})}$$

   where $Cat(x, [p_1, \ldots, p_m])$ is a categorical PMF of x with parameters $p_1, \ldots, p_m$. Like in TGAN, this results in a probability distribution over the components m_i.

3. Sample $k_{i,j} \sim Cat(k; [\beta_{i,j}^{(k)}]_{k=1\ldots m})$ and convert it to a one-hot representation $\beta_{i,j}$.

4. As a final step, just as it happened in TGAN, the data $c_{i,j}$ is normalized with the mean and standard deviation of the most probable Gaussian, following $a_{i,j} = (c_{i,j} - \mu_i^{(k)})/4\phi_i^{(k)}$. The value is then clipped to $[-1, 1]$ which covers 99.99% of all samples. Value $c_{i,j}$ is then represented by $a_{i,j}$ and $\beta_{i,j}$.

Three other methods are proposed to improve training. The first one is a *conditional vector*, which allows for conditioning on a certain value of a certain column through one-hot encoding. Without going into specifics, this method helps broadening the variety of samples by sometimes explicitly conditioning on a specific value of a column, forcing the generator G to mimic this. Normally, if a columns contains a class that spans the majority (say $> 90\%$), it is easy or the generator to ignore this value because the critic does not enforce having the minor class present. With this explicit conditioning, G will be punished if it deviates from the required class. This conditional vector consists of all the categorical columns one-hot encoded concatenated together, where the only 1 is at the desired class. So say there are two columns, $D_1 = 1, 2$ and $D_2 = 1, 2, 3$ and the condition is $(D_1 = 2)$. The columns are then represented by their one-hot encoded vectors $\mathbf{m}_1 = [0, 1]$ and $\mathbf{m}_2 = [0, 0, 0]$ which are combined into $cond = [0, 1, 0, 0, 0]$.

This also relates to the second contribution: they add a section to the loss function which is the cross entropy between the *cond* and the generated one-hot encoding of the categorical columns.

The last proposition closely ties together with the previous two, namely how does one determine on which class to condition and with what probability. This is done by first selecting one of the categorical column with equal probability. Compute the PMF of the values in that column where the mass of each value is the logarithm of its frequency in that column. As a last step, select a random value from the PMF and set this value to 1. Set all other values of the conditional vector to 0: $cond = \mathbf{m}_1 \oplus \ldots \oplus \mathbf{m}_N$ where N is the total number of categorical columns, $\oplus$ means the concatenation of two vectors and the selected value in the selected columns is 1.

The architecture of the network has also changed. The authors found the LSTM cells were not required to achieve the required performance, so the network is now just some dense layers with batch normalization [30], which improved training time significantly. Additionally, although it isn't mentioned anywhere in the paper, they changed the GAN architecture from the vanilla GAN to the WGAN-GP basis.

This paper is interesting because it is the first paper that compares against other GAN based data synthesis methods. They compare against TableGAN, MedGAN and some others. They do not, however, compare against their own previous TGAN model. Interestingly

enough, in both they have some overlap in evaluation, for example, F1 scores on the Census dataset, where their older TGAN model outperforms their newer CTGAN model. The other metric, accuracy on the covertype dataset, does show improved results for CTGAN, so we would have really liked to see TGAN be incorporated into the results of CTGAN.

The expectation of CTGAN was quite high, but unfortunately the code provided by the authors did not work in our testing. As a consequence, we did not include CTGAN in our comparisons and used TGAN to improve upon.

4.2 Baselines

One of the major gripes with such a young area of research is that evaluations are not always comparable, due to different datasets and different metrics. Additionally, evaluating synthesized data is hard in and on itself due to the use of different definitions of what is a "good" sample, especially with tabular data.

In the literature, many different baselines are used, which makes comparing different deep learning based generative models surprisingly difficult; the reported results do not compare with each other. In general, publications on this area have reported on statistical models and anonymization tools as baselines. Even though these baselines are not meaningless, the omission of other deep learning based generative methods is a major issue. TableGAN does compare against DCGAN, but this mainly indicates the effect of the adaptations they made to create TableGAN, and not as a competitive tabular data synthesis model.

4.2.1 Statistical Generation Techniques

In the statistical models we essentially include everything that does not fall into neural generation techniques nor data perturbation and anonimization tools. Statistical generation techniques include graphical models like Bayesian networks, but in this case also includes applying random noise to real data points to get permuted data points and independent sampling, which is only possible for binary values. Finally, Gaussian copula are also included in this section.

Bayesian networks. Bayesian networks are generally quite capable of modeling relationships, but struggle when datasets get larger. In the evaluation TGAN, Bayesian networks are applied to model both the column independently as well as to the whole dataset. The first approach does not perform well, since it does not capture any relations between columns. During evaluation, many machine learning models applied to these samples deteriorate into predicting the majority, which is a consequence of the absence of inter-column relations. The second approach takes an extremely long time to train, even with a subset of the dataset. In CTGAN, they use two slightly different approaches, of which one is quite old [13], while the other is more recent and also targets privacy in Bayesian networks [60]. While these baselines at least give some usable results, they are still significantly outclassed by most of the deep generative methods.

Random noise. Random noise is applied by randomly switching binary values with 10% probability. This approach is straight forward and has the expected results. Distributions change somewhat, while F1 scores correlate a bit less between the datasets.

Independent sampling. Independent sampling means a Bernoulli distribution modeled by the data distribution. So, as expected, this achieves the exact same distribution for binary features as the real data but is not feasible for any other datatype.

Gaussian Copula. Without going into exactly how Gaussian copulas work, they allow the transformation of any distribution to a Gaussian distribution, and create an encoding for the original values in the new domain. After converting all rows to this encoding, the covariance matrix Σ can be computed. The parameters for the distributions of each column, together with Σ become the generative model for the dataset. This modeling is done per column, meaning that this approach also is not able to capture relations between columns [58, 51].

4.2.2 Neural Generation Techniques

Neural generation techniques are a bit more interesting, and include Restricted Boltzmann Machines (RBMs) and Variational Autoencoders (VAEs). Additionally, in [59] they adapt a VAE to be able to handle different data types and provide quite a good comparison. However, the RBM is only used for binary variables, meaning its utility is quite limited. The ins and outs of the VAE have been discussed in section 3.7, and should make some of the limitation clear with respect to this evaluation. Without modifications, it is only applicable on binary and categorical data. This is why the adaptations done by [59] are quite interesting and provide state-of-the-art results in some of the evaluated tasks.

4.2.3 Data perturbation and anonimization tools

TableGAN is the main contributor to this section, and compare their approach with two data anonymization tools: ARX [1] and sdcMicro [57]. Both are open source tools, of which sdcMicro is also an R package. Both packages have different functionalities and options with regard to data perturbation and privacy. Since these baselines will not come back in this thesis, we will not discuss these any further. For more information, refer to [50].

4.3 Evaluation metrics and methods

In this section, we will outline the different evaluation methods that have been applied so far, as well as discuss their advantages and disadvantages. The criteria listed below are all applied in a situation where we have a trained generator G from which a number of data points are sampled. In this thesis, we propose a combination of some of these methods and several novel methods to form a cohesive metric for the evaluation of synthetic data.

In short, the three views on different evaluation are:

1. Statistical properties
2. Machine learning efficacy
3. Privacy

4.3.1 Statistical properties

Statistical properties refer to very basic properties of the data, like the mean and the standard deviation. These kinds of properties are basic enough to have their similarity

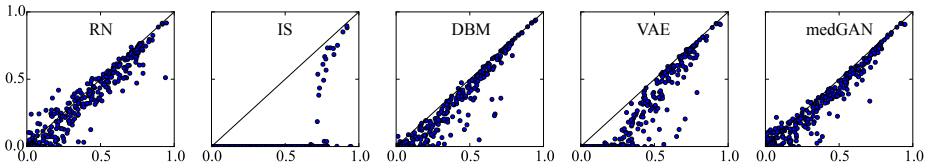


Figure 4.4: F1 scores for predictions on all columns on all datasets. From MedGAN [12]

be a minimal requirement for each data synthesis method. Additional metrics can also be calculated over these properties, like distance between correlation matrices and differences in mean and deviation, for a more quantitative way of evaluation.

4.3.2 Machine learning efficacy

Machine learning efficacy looks at how usable data is for cross domain applications. Additionally, it gives insight into some relations that might be more or less apparent in on of the two datasets. This metric is one of the recurring evaluation methods in the related work. Typically, it is done by comparing a classification metric (F1-score, Accuracy) or a regression metric (Root Mean Squared Error, Mean Absolute Error) of different models on both the synthetic and the real dataset. For example, TableGAN reports the F1 and Mean Absolute Percentage Error for applicable column from their test datasets. A very interesting twist to these metrics is when a model trained on real data is evaluated on synthetic data and vice versa. This really shows whether a model would be intercompatible between datasets. These evaluations are usually visualized as in figure 4.4.

4.3.3 Privacy

Privacy is a bit more controversial as a metric, with very wide ranging methods of evaluation. Some papers do nothing with it during evaluation [59], while others use it as their main focus [50]. For example, in [50, 58] they calculate the mean distance between each synthetic data point and its closest data point from the real dataset. Some also go a lot further, such as [50] attempting a membership attack, which is a specific attack to machine learning models to distinguish values that appeared in the real dataset. Additionally, [12] dives into whether one can recover a real data point if some of the values of that data point are known, which is a very interesting approach. Luckily, they find that this does not matter a great deal which inspires some confidence in the ability of these models.

4.3.4 Human Experts

Human experts have been asked to evaluate generated data in Choi et al. [12]. In one of their experiments, they let a medical doctor rank 100 records on realness, which consisted of 50 real and 50 fake records. In their evaluation they find that the majority of samples that were classified as not very real had some conflicting information in them, like a woman with prostate cancer.

Chapter 5

Proposed Methods

In this chapter, we elaborate on our three proposals. The first two are adaptations to the architecture of TGAN, which we hypothesized perform better. The first proposal reflects a major trend in neural networks, where so called skip-connection are added to a network to increase gradient flow and information retention. The other is a GAN specific adaptation, the WGAN-GP architecture, which uses a different training setup and loss function that reduces mode-collapse and has a more interpretable loss function. Our third proposal is a similarity metric to evaluate synthetic data against real data, which captures the total similarity in a single value and displays a more balanced view of how they compare. The two proposed adaptations are different versions of a GAN synthesizer, while the evaluation metric is in the similarity score section. In figure 5.1, we describe an overview of the framework, along with where our proposals fit into it.

5.1 GAN Synthesizer

5.1.1 TGAN-skip

The first proposal is adding skip connections to TGAN, which will applicably be called TGAN-skip. Skip connections have been proven to be extremely effective in discriminative models like ResNet [24] and DenseNet [28]. The main advantage of adding skip connections

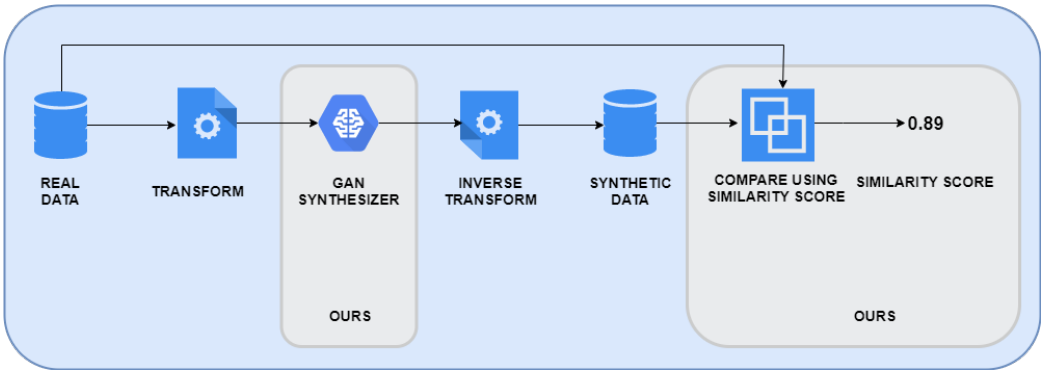


Figure 5.1: The framework in which we do our proposals. We propose two versions of the GAN synthesizer, TGAN-skip and TGAN-WGAN-GP. Our proposed metric, the Similarity Score, is in the scoring part.

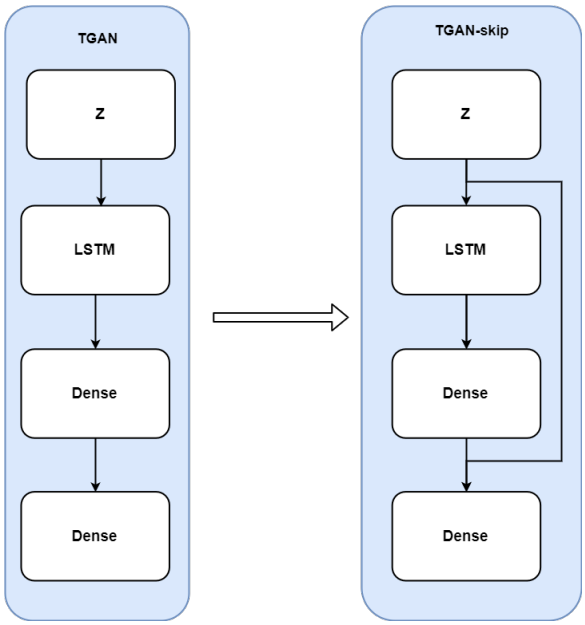


Figure 5.2: The adaptation made to TGAN to get TGAN-skip. The skip connection is between the noise input z and the second dense layer.

is that this solves the vanishing gradient problem. The vanishing gradient problem occurs in models with many layers where the activations in later layers start to near 0. With these low activations, gradients are also very small and take a long time to converge. Skip connection allow earlier activations to skip over layers to keep the magnitude of the activation high. With higher activations come higher gradients and so we can train deeper models (with more expressive power).

Additionally, skip connections retain old information, which is used in models like U-net, a semantic segmentation network. In these models, data is reduced to a very low dimension after which it is scaled up again. Keeping the exact details of images is very hard this way. To help the model do this, old information is combined with new information to finish at the desired results.

Skip connection are already used in the generator of MedGAN [12], where they claim it improves training. To understand the changes made to TGAN, I will first detail the architecture a bit more. As stated in section 4.1.1, TGAN's main components is an LSTM cell that loops over the columns. The output of the LSTM cell is not directly used as output for the whole, but rather goes through two dense layers before creating the final output. Additionally, the output of these two layers goes through another dense layer to formulate the attention vector for the next LSTM iteration, where it is used as input to the LSTM cell. The skip connection is added between the first two dense layers. The hypothesis is that this decreases convergence time due to greater gradients. Additionally, it should yield higher overall performance due to the extra information that is able to flow through. So the resulting model would be one that trains faster and achieves greater performance than it would otherwise do.

To make it concrete, the adaptation done is as follows. Between the two fully connected layers, we concatenate the input z and output of the first dense layer. Note that in the

actual code, this change is done three times due to different handling of categorical values and the two parts of continuous values. The following piece of code

```
    z = tf.placeholder_with_default(z, [None, self.z_dim], name='z')
    ...
    h = FullyConnected('FC', output, self.num_gen_feature, nl=tf.tanh)
    w = FullyConnected('FC2', h, gaussian_components, nl=tf.nn.softmax)
    outputs.append(w)
```

becomes

```
    z = tf.placeholder_with_default(z, [None, self.z_dim], name='z')
    ...
    h = FullyConnected('FC', output, self.num_gen_feature, nl=tf.tanh)
    h = tf.concat([h, z], axis=1)
    w = FullyConnected('FC2', h, gaussian_components, nl=tf.nn.softmax)
    outputs.append(w)
```

5.1.2 TGAN-WGAN-GP

The original TGAN architecture uses the vanilla GAN training, except for an extra KL-divergence term in the loss of the generator. With the WGAN-GP architecture outperforming most contenders in many GAN related tasks, the hypothesis is that also in this task, a WGAN-GP approach will improve results. For the exact details on why this training method has improved performance over the vanilla GAN, we refer you to section 3.10. In short, the following effects should occur: WGAN-GP has a loss function that has semantic meaning (i.e. decrease of the discriminator loss means a better generator), which is not the case with a vanilla GAN. In the vanilla GAN, the loss can increase, even when sample quality is increasing. Additionally, this loss function does not suffer from the caveats of the vanilla GAN, where an imbalanced generator and discriminator caused training stagnation and finally, using the Wasserstein distance should reduce the occurrence of mode collapse. In total, the model should converge smoother and faster to the optimum, as well as have improved performance over vanilla GAN.

While WGAN-GP conceptually is a small change, it requires some intrusive alterations to the code. First, the loss function must be adapted to the Wasserstein distance and the gradient penalty must be calculated. The batch normalization in the discriminator must be adapted to layer normalization. Batch normalization in the discriminator has many strange artefacts when combined with a Wasserstein distance metric, as discussed in section 3.10. Additionally, the parameters of Adam must be changed to reflect the changes of [23, 5] ($\beta_1 = 0$, $\beta_2 = 0.9$), essentially transforming Adam into RMSprop. The sigmoid activation on the last layer needs to be removed, and lastly, we need to change the ratio of training between the generator and discriminator. In WGAN-GP, the discriminator trains more iterations than the generator, in a $5 : 1$ ratio, respectively.

5.2 Similarity Score

To improve the evaluation methods of synthesized data, we propose our own metric, called the **similarity score** SS. This score aggregates different measures to cover all aspects of data. The goal is to get a single value representing how close a synthetic dataset is to a real dataset. For this score, we focus on statistical and machine learning metrics. Apart from the single resulting value, some additional metrics are reported to highlight certain aspects of the datasets. In this chapter, we detail what features are weighted and how, and it should give a sense of what characteristics of a synthetic dataset are most important for this goal.

If the datasets being compared are very alike, the scores of the metrics should be very similar for each metric. If the datasets are the same, the scores should be the same and thus we can benchmark the similarity by doing a correlation analysis on the results of most metrics. Pearson's correlation coefficient is in this case quite well suited because Pearson's correlation coefficient is a metric that determines the linearity between two variables and in this case, the optimal relation is linear. One disadvantage of this is that a linear relation need not have a slope of 1, but the chances of encountering a linear relationship with a slope other than 1 are quite low. This is mainly because the synthesizers are trying to replicate relations from the real data. If the metrics applied to the synthetic data have a linear relationship with a slope other than 1, the synthesizer somehow modeled the relations with some scalar consistently.

Alternatively, one could argue using a rank correlation coefficient like Kendall's τ. The argument would be that if some metric u achieves better results on dataset A than on dataset B, another metric v would do so as well. Applied to this situation, Pearson's measures how close the results of the estimators are to each other, contrary to consistency between estimators. The problem with just comparing the ordering is that it does not matter how much higher or lower a value is. Due to this characteristic, we take Pearson's correlation metric in most of the upcoming evaluations.

The results of these metrics generally range between $[-1, 1]$ and indicates the correlation coefficient where -1 means a negative correlation, 0 means no correlation and 1 means positive correlation. A negative correlation means that when variable a goes up, variable b goes down and vice versa.

5.2.1 Similarity Score - Statistical Measures

Basic measures

The statistical evaluation starts with aggregating four simple statistics, which are the mean, median, standard deviation and deviance. One could argue that some metrics do not really add information in most cases, but there is not downside to including them, and it does stabilize the results by having more data points. This gives four values per column per dataset. We calculate the correlation coefficient for the similarity score. However, because some columns can contain very large numbers while other columns can be very small, a single large columns can impact the Pearson's correlation coefficient over the whole dataset significantly. To counter this effect somewhat, we use the *Spearman's* ρ [16] correlation instead of Pearson's. Spearman's ρ is similar to Kendall's τ in that it is a rank correlation metric but also takes into account the values. In practise, it determines whether there is a monotonic relationship instead of a linear one (Pearson's). The result is a metric similar to Pearson's, but less perceptive to outliers in the tails. Applying Spearman's ρ gives the first

value of our similarity score: S_{basic}. This is calculated by

$$S_{basic}(A, B) = \rho_{spearman}(\langle mean(R),\ std(R)\rangle, \langle mean(F),\ std(F)\rangle) \tag{5.1}$$

where $\langle A, B\rangle$ denotes the concatenation of two lists and $mean(A)$ and $std(A)$ refer to the list of means and standard deviations of each column of table A, respectively.

An additional sanity check will be to report the number of duplicate rows in the real dataset and each of the synthesized datasets. This can give us an idea whether mode collapse occurred and how well distributed the data is originally.

Correlation matrix for mixed data types

The second statistical evaluation considers the correlations between the columns of each dataset. Building a correlation matrix for a table is important to understand the relationships between the columns. This approach gives us $2N^2$ values in total, where N is the number of columns in the datasets.

Getting a correlation matrix is a bit more challenging than usual, because we have non-continuous data. Since correlation has a specific mathematical definition, we will refer to cases with possible non-continuous situations as *associations* instead of correlations. The correlation matrix of a table with mixed data will then be called the *association matrix*. The range for associations with mixed data or purely categorical data is often $[0, 1]$ where 0 and 1 have the same meaning as in the correlation case with Pearson's. With categorical values however, a negative association cannot exist because for a negative correlation to exist, both variables are required to have an ordering. With our categorical data (often ordinal is also considered categorical in these architectures), there is no order and thus there cannot be a negative association between non-continuous data. The difference in range is recognized, but no normalization is applied because the value 0 means no correlation in all cases. So without further ado, lets introduce the methods to calculate these associations.

Continuous - Continuous. The case of continuous - continuous is easy, because it allows us to use Pearson's correlation. This metric is also able to measure negative correlation between columns. We revert back to using Pearson's instead of Spearman's because the impact of outliers will be much lower, due to the size of our datasets.

Continuous - categorical. In the case where one column is categorical and one continuous, we take the *correlation ratio* [18], another measure introduced by Karl Pearson, the same that proposed Pearson's correlation coefficient. Mathematically, it is defined as the weighted variance of the mean of each category divided by the variance of all samples. In short it answer the question: given a continuous value, how well we can tell which category it belongs to. The range of the correlation ratio is $[0, 1]$.

categorical - categorical. In the case of two categorical columns, we use *Theil's U* [54][1] as a metric. Also called the *Uncertainty Coefficient*, Theil's U is based on the conditional entropy between two variables. In short, it is a measure of how many values and with what probabilities there are for variable x, given a value for variable y. The range for Theil's U is also $[0, 1]$. An alternative would be to use Cramer's V. However, Cramer's V is symmetric, and thus fails to capture asymmetries between columns. Theil's U is able to capture this

[1] Shannon [54] uses the term uncertainty coefficient

and thus preferred.

By using these metrics when applicable, we can create an association matrix for a tabular dataset. The logic of comparing metrics of datasets by linearity can be reused here, and thus we calculate the correlation between the datasets. If this is 1, we know that the column associations are very similar between the datasets. Before doing the regression, the diagonal values are removed, since it indicates the correlation/association of a column with itself, which will always be 1. Because this gives no additional information and gives the correlation a bias towards 1, we remove these values. The correlation coefficient gives us the second value for the similarity score S_{corr}. This is calculated as

$$S_{corr}(A, B) = \rho_{pearson}(association(A),\ association(B)) \tag{5.2}$$

where *association* refers to the association matrix of each table.

For completeness, the distance between these matrices is also calculated on an element-wise basis using the mean absolute error and root mean squared error. However, these distances are reported as is and are not taken into account for the similarity score under the assumption that this information is incorporated in the correlation coefficient.

Mirror Column Associations

In the previous section we only calculated associations within a single table. However, it is also quite interesting to see if a column in dataset A correlates with that same column in dataset B. For this, we calculate the association values the same way as in the previous section. The results in N values, where N is the number of columns. Because these values are already a comparison between the datasets we simply take the mean as final value for the similarity score: S_{mirr}. This measure will be named the mirror column associations. Formally, this is as follows

$$S_{mirr}(A, B) = \frac{1}{N} \sum_{i=1}^{N} association(A_i, B_i) \tag{5.3}$$

where N is the number of columns in A and B and A_i means the ith column of A.

Principle Component Analysis

Principle component analysis (PCA) is a well known statistical method of dimensionality reduction while maintaining the same amount of information. It tries to convert the data onto principal components, which are linearly uncorrelated vectors. These are then sorted by the amount of variance captured in that component, of which we keep the top k values. If k is smaller than the original number of dimensions, the dimensionality is reduced.

In this case, we take explained variance of the top five principle components. This approach has three problems for our normal evaluation. First, PCA is quite similar between datasets where other metrics report some differences. Additionally, we only take the top five components, which is not that much when doing a correlation analysis. Lastly, the same problem as with the basic statistical features occurs again. When there is a large magnitude difference between columns, the first PCA components capture the variance of columns with a larger magnitude more. The small number of data points makes using

the Pearson's hard and thus we resort to something else. We calculate the mean absolute percentage error, indicating the average error in percentages, as follows:

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{y_i} \tag{5.4}$$

where N is the total number of values being compared, i.e. the number of principle components.

With pc_A indicating the explained variance of the principle components of dataset A and N being the number of principle components that are evaluated, we calculate the final value for the similarity score with

$$S_{pca}(A, B) = 1 - \frac{MAPE(\log(\text{pc}_A), \log(\text{pc}_B))}{100}$$

We apply the log to circumvent having very large percentage errors ($> 1000\%$), which swing the similarity scores quite a lot. This gives us our fourth value for the similarity score: S_{pca}.

5.2.2 Similarity Score - Machine learning efficacy

The machine learning measures are split between analysis on continuous and categorical columns. The classifiers and regressors used are chosen due to their common usage and not for any specific performance on these datasets. If the target column is categorical, the following classifiers (all from scikit-learn) are used. If no parameters are specified, the default parameters are used:

1. `Logistic Regression (multi_class = auto, solver=lbfgs, max_iter=500)`
2. `Random Forest`
3. `Decision Tree`
4. `Multilayer Perceptron` ($[50, 50]$, `solver=adam, activation=relu, lr=adaptive)`

If the target column is continuous, the following regressors are used:

1. `Random Forest (n_estimators=20, max_depth=5)`
2. `Lasso Regression`
3. `Ridge Regression (alpha=1.0)`
4. `ElasticNet`

For evaluation of the effectiveness of the trained classifiers, the F1-score is used. The F1-score is the harmonic mean between the precision and recall, two other measures for model performance. In a multi class situation, F1-scores can be applied in different ways, but we take the `micro` version, where all evaluations are done global instead of within a class.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{5.5}$$

However, when working with continuous data we use the root mean squared error (RMSE) to evaluate the models results. As the name indicates, the RMSE is calculated as follows

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2} \tag{5.6}$$

where N is the number of results that are compared.

The method of evaluation for both is similar, so we will refer to whichever is chosen as the estimators from now on. Two sets of estimators are created (E_R and E_F) and the steps of the evaluation is described below.

1. Real data R and fake data F are split in a 80%/20% split for train/test and the target column is sliced of, resulting in F^x_{train} and F^y_{train} for the fake training set and similar for the test set and the real data.

2. E_R is trained on R_{train} and E_F is trained on F_{train}. We now have one set of estimators fit on the real training data and one set of estimators fit on the fake train data.

3. Taking the evaluation of E_R as an example: we evaluate E_R on both R_{test} and F_{test} with the appropriate metric (F1/RMSE). We do the same for E_F. This results in two scores for each estimator.

4. If we have F1 scores, we calculate MAPE on the resulting scores to get single value and take $1 - MAPE$. If we have RMSE scores, we apply Pearson's to get a correlation coefficient. This difference is due to correlations becoming arbitrary when the points are clustered together closely, which happens when the real and fake dataset are quite alike and the F1 scores only differ ±1%. The results is the fourth value for the similarity score: S_{est}. Formally, this is

$$S^{regr}_{est} = \rho_{pearson}(RMSE(E_R),\ RMSE(E_F)) \tag{5.7}$$

$$S^{class}_{est} = 1 - MAPE(F1(E_R),\ F1(E_F)) \tag{5.8}$$

where $RMSE$ and $F1$ return the scores of the given estimators on both test sets.

5.2.3 Privacy evaluations

These results are not part of the similarity score, but are relevant metrics to report on. With regards to privacy, two metrics are used. The first is a simple analysis whether any rows in the synthetic dataset are identical to corresponding rows in the real dataset. Generally, this is not desired and possible regularization or checks might be required to prevent this. The second metric is the mean and standard deviation of the distance between each fake record and the most similar real record. This is a metric also used in related work [58, 50], and offers some perspective into how similar individual records are between datasets. The desired outcome is a high mean with a low standard deviation. If the standard deviation is high (or close to the mean), it could mean that many synthetic records are quite similar to the real records. This metric is calculated by scaling all continuous values to $[0, 1]$ and one-hot encoding the categorical values. We then take the Euclidean distance between each row. The Euclidean distance has been a controversial metric for high dimensional data, and results in almost arbitrary values in those high dimensions. In this work we have chosen to use the Euclidean distance to be able to compare it to related work, which also uses this metric.

5.2.4 Visual evaluation

Visuals are a very powerful tool for humans to verify results and recognize patterns. The visual evaluation in this framework uses the following parts:

1. Columnwise mean and standard deviation
2. Cumulative Sums
3. Column Correlations

Columnwise mean and standard deviation

This is not an advanced metric and does not reveal any hidden relations, but functions as a quick sanity check. The means and standard deviations of each column are plotted on a log scale. If the plotted values follow the diagonal, the data has comparable means and standard deviations. Furthermore, the general results do appear to have a correlation with how well these values follow the diagonal, meaning that if many points deviate from the diagonal, it is likely the cumulative distributions and quantitative measures will follow a similar pattern of deviations.

Cumulative Sum

To visually inspect the similarity between the distributions per column, we plot the cumulative sum of each column for both the real and the fake data on top of each other. This gives one a quite thorough understanding of a column with just one plot, and works for both categorical and continuous columns. Note that this plot does not give any insights into the relations between columns, giving it limited representational power for the whole table. It does allow one to determine which kind of values and columns are easier or more difficult than others.

Column Correlations

The third visualization shows one an association table for the both the real and synthetic data. It gives a clear understanding of what columns have associations with each other, and shows where the synthetic data diverges, indicating struggles that the model had with modeling this relationship.

5.2.5 Similarity score

From these different evaluations metrics we have now obtained five values used for the similarity score SS: S_{basic}, S_{corr}, S_{mirr}, S_{pca} and S_{est}. The similarity score is then calculated by taking the mean of these five values.

$$SS = \frac{1}{N_S} \sum_{s \in S} s$$

where S is the set of all similarity metrics $\{S_{basic}, S_{corr}, S_{mirr}, S_{pca}, S_{est}\}$ and N_S the length of S.

Chapter 6

Experimental Setup

To evaluate our proposal, we perform an experiment with five models (TGAN, TGAN-skip, TGAN-WGAN-GP, MedGAN, TableGAN), and three publicly available datasets. For evaluation, we use our proposed method outlined in section 5.2 to benchmark the models.

6.1 Data

We select three datasets for evaluation, which are the Census dataset [40, 17], the Berka Czech Financial dataset [7] and Creditcard Fraud dataset [2]. These datasets are chosen for their differences in data type distributions. The census dataset is mostly categorical values, the creditcard dataset is mostly continuous values and the Berka dataset is a nice mix of both. This allows us to detect whether the models work better or worse for certain kinds of data.

The Berka dataset is a Czech financial dataset from a bank, including transactions, bank accounts and more. The Census dataset is a sample from the US census, used to predict whether someone earns over 50k per year. The creditcard dataset is a fraud prediction dataset, where the features are the top 28 PCA features of some unknown financial input. In table 6.1, we show some statistics of these datasets.

Dataset	#Features	#D	#C	#Rows	#Labels
Berka [40, 17]	8	**4**	**4**	1056320	3
Census [7]	40	**33**	7	199522	2
Creditcard [2]	30	1	**29**	248808	2

Table 6.1: Properties of each dataset. #D indicates the number of discrete/categorical columns, #C the number of continuous columns.

6.2 Implementation details

We used the same data transformation steps as TGAN, to get comparable results. In short, each continuous value is represented by two values, to which Gaussian distribution it most likely belongs and where in that Gaussian the value relatively lies. Each categorical values is onehot-encoded. For exact details, please see 4.1.1.

Our LSTM cell has a hidden state of length 50, which an output feature length of 64. The learning rate is 0.001 and our discriminator has one dense layer with 100 nodes. We use a L2-norm of 0.00001 and our random noise vector has a length of 200.

All of these models were trained for 100 epochs on the datasets, where the number of steps is $N/batchsize$ where N is the number of rows in the dataset and $batchsize = 200$. After training, we sample $100k$ rows for evaluation. Earlier work did evaluations on much smaller datasets, with 10k rows or less. In our experiments, this did not provide stable results for most of the tests, specifically when evaluated on machine learning models where accuracy and F1 scores could swing multiple percent points between runs when using these small datasets.

6.3 Experiment 1

Our experiment will be performed by generating data with five models: TGAN, TGAN-skip (ours), TGAN-WGAN-GP (ours), MedGAN and TableGAN. CTGAN is a notable omission, because in our testing, their model did not work, even on the toy datasets provided by the authors. The five resulting synthetic datasets are then evaluated using our proposed method: the similarity score. With this experiment, we show that our proposed models perform better than alternative models and our proposed evaluation metric correlates well with sample quality and is a good indication of synthesizer performance. Because our evaluation method is an aggregate from several from smaller parts that occur in the literature, we can show for each part that our models perform better.

Chapter 7

Results: Experiment 1

The results of the previously mentioned evaluations will be discussed, while highlighting the weaknesses and strengths of the individual models. Not all visuals will be presented in the results section, but remaining ones can be found in appendix A.1.

7.1 Basic statistics

Starting with the first evaluation, we take a look at the mean and standard deviations of the real and fake dataset. We plot the log transformed values of all the *numeric* columns. The assumption is made that if a synthesizer already fails to capture these basic properties, more derived features will likely suffer the same fate. In the Berka dataset (figure 7.1) we observe that four of the five synthesizers capture these properties with relative ease. MedGAN, however, already has a hard time reproducing these values, which is not a great sign. This behaviour is continued in the Census dataset.

In figure 7.2 with the Creditcard dataset, more differences can be noticed. Most models are able to capture the standard deviations, but have quite a hard time capturing the correct means. It seems the synthetic data differs by multiple orders of magnitude from the real data, but keeps the standard deviations in the same order of magnitude. It is clear that this dataset was much harder to synthesize.The number of duplicate rows in the datasets are presented in table 7.1. The quantitative results can be found in appendix A.1

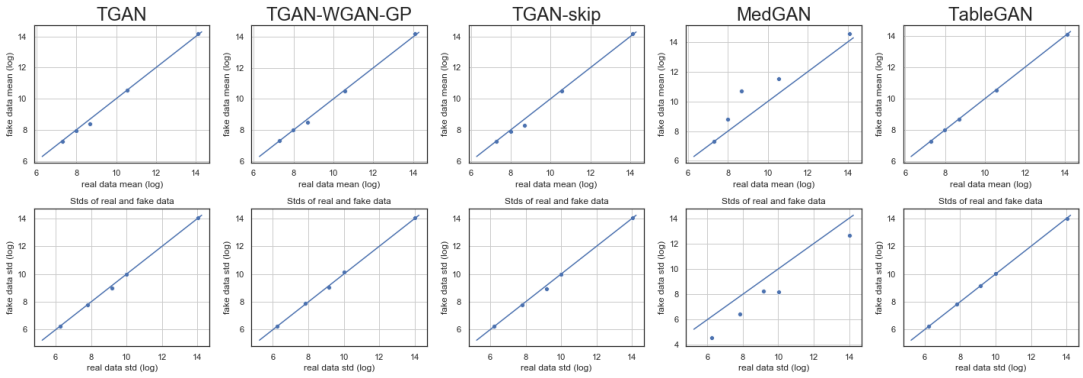


Figure 7.1: Mean and standard deviations of each column of the real and synthetic Berka dataset. All values are log transformed.

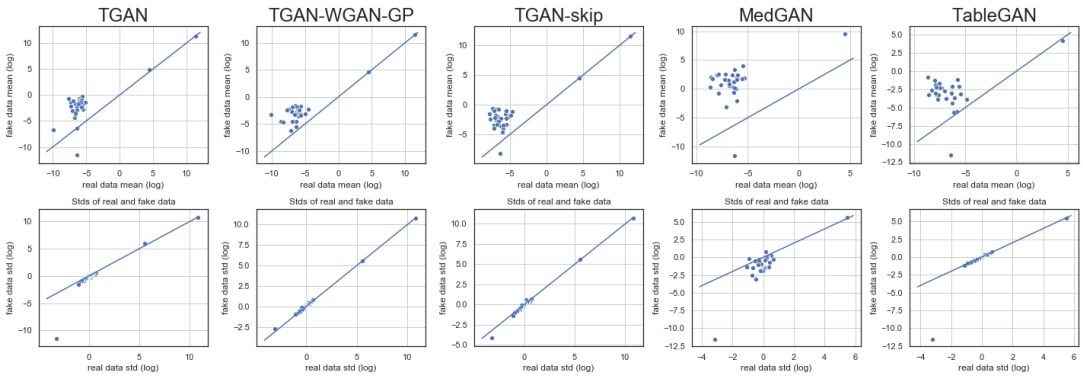


Figure 7.2: Mean and standard deviations of each column of the real and synthetic creditcard dataset. All values are log transformed. Recall the line follows the $x = y$ diagonal.

	Real	TGAN	WGAN	SKIP	MedGAN	TableGAN
Berka	0	0	0	0	7563	0
Census	24263	27668	18481	8472	23115	3356
Creditcard	3881	0	0	0	212	0

Table 7.1: Number of duplicate rows per dataset per synthesizer. Using the real creditcard dataset as an example, there are not 3881 identical rows, but rather there are 3881 rows that occur twice or more.

7.2 Column Correlations

The difference in correlation values plot is capped at 0.3, since we consider this a large deviation from the target. However, this max is not present when calculating distances and correlation values. Figure 7.3 shows the column wise correlations of the Berka dataset. It becomes clear TableGAN is lacking behind the TGAN versions, and has a hard time capturing some correlations. The later columns of Berka contain the categorical columns, which seems to be an obstacle for TableGAN. Also, we see the performance of MedGAN is very poor, being essentially random. In figure 7.4, a similar pattern appears with TGAN-WGAN-GP seemingly performing best. TGAN and TGAN-skip show similar pattern and are not much worse. TableGAN does not seem to perform necessarily worse, but shows a different pattern with regards to what is was and was not able to capture. MedGAN shows no capability to capture results. This pattern repeats for the creditcard dataset. The quantitative results are presented in table 7.2, where we can see the same pattern as just described being reproduced. The vanilla TGAN takes the top spot in the Berka dataset, while TGAN-WGAN-GP is best in class with the census and creditcard datasets.

7.3 Mirror Column Associations

The mirror column associations' quantitative results are presented in table 7.3. We observe a result close to the column associations, where TGAN performs best on the Berka dataset, but is outshined in the other synthesization tasks. TGAN-WGAN-GP is the best in 2/3

	TGAN		WGAN		SKIP		MedGAN		TableGAN	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
Berka	**0.0244**	**0.0184**	0.0883	0.0619	0.0407	0.0288	0.2251	0.1781	0.2131	0.1284
Census	0.0674	0.0295	**0.0488**	**0.0218**	0.1057	0.0479	0.4341	0.2801	0.0865	0.0364
Creditcard	0.0786	0.0525	**0.0540**	**0.0357**	*0.0681*	*0.0459*	0.5617	0.4861	*0.0579*	*0.0381*

Table 7.2: Correlation matrix distances. Bold indicates the best in row, while italics indicates an improvement over TGAN. The WGAN-GP approach outmatches TGAN in 2/3 tasks, while TGAN-skip improves in 1/3. TableGAN improves on TGAN with the creditcard dataset, but otherwise severely lacks behind TGAN. MedGAN does not improve on TGAN.

Figure 7.3: Associations per column from the real dataset and each of the synthesizers. The bottom row is the absolute difference between the synthesizer and the ground truth. The range of the difference is reduced to $[0, 0.3]$.

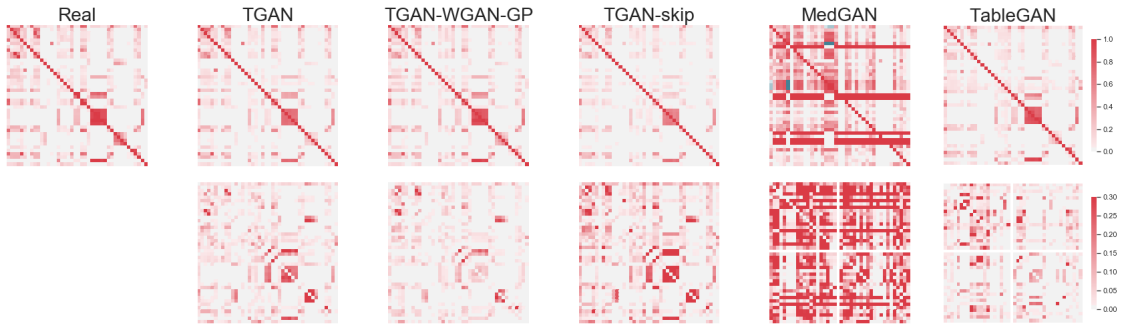


Figure 7.4: Associations per column from the real dataset and each of the synthesizers on the Census dataset. The bottom row is the absolute difference between the synthesizer and the ground truth. The range of the difference is reduced to $[0, 0.3]$.

	TGAN	WGAN	SKIP	MedGAN	TableGAN
Berka	**0.9956**	0.9594	0.9939	0.7654	0.6283
Census	0.9574	**0.9769**	0.9051	0.0645	0.9129
Creditcard	0.4541	**0.6629**	*0.4594*	-0.0634	*0.5608*

Table 7.3: Mirror columns association results for each synthesizer and dataset. Values are means of associations. Higher is better. Bold indicates best in row, while italics indicate improvement over TGAN.

tasks, with TGAN-skip and TableGAN improving upon TGAN in the creditcard tasks. MedGAN does not improve on TGAN in any task. A pattern seems to become clear where TGAN is best in the Berka dataset but not in others. A possible explanation is that the Berka dataset has many more data points (5x and 4x more than census and creditcard, respectively). The hypothesis of the TGAN adaptations was that they would improve convergence time, and it could be the case that TGAN has not yet converged fully after the 100 epochs.

7.4 PCA Variance Correlations

Table 7.4 presents the PCA correlation values for each synthesizer and dataset. Here we see that TableGAN takes the top spot in every task. Between the TGAN versions, we see the TGAN is best for the Berka dataset and the creditcard dataset, while WGAN is better on the census dataset. TGAN-skip improves upon TGAN only in the census dataset. MedGAN is the worst in every task. The sudden performance of TableGAN is caused by the fact that PCA is meant for continuous data and does not accurately reflect the synthesizer performance on categorical data. Additionally, the magnitude of continuous data is typically much larger than that of categorical data, which is often one-hot encoded. In short, PCA prioritizes performance on continuous columns, and as we will see soon, TableGAN is very good in approximating individual column distribution.

In hindsight, PCA alone might not be the best metric to evaluate a mixed data set with. Creating a combined metric with *multiple correspondence analysis* [22] could have provided a more balanced representation of the datasets. At the moment, this metric is the only biased metric for a specific data type and it impacts the final results somewhat because TGAN and its variants perform worse on continuous data than TableGAN.

7.5 Estimator efficacy

Things get quite interesting in this evaluation step. Both Berka and the census dataset are approached as a classification problem, predicting the transaction type and income bucket, respectively. However, in the creditcard dataset the number of fraudulent transactions is very low (0.1%). As a consequence, some synthesizers do not generate records belonging to this very minor class at all, like TGAN, MedGAN and TableGAN. This prohibits us from evaluating these datasets with classifiers, because they only have one class. The alternate

	TGAN	WGAN	SKIP	MedGAN	TableGAN
Berka	0.9456	0.9399	0.9424	0.8236	**0.9465**
Census	0.9507	*0.9739*	*0.9643*	0.6907	**0.9748**
Creditcard	0.8501	0.7810	0.7266	0.1726	**0.8667**

Table 7.4: PCA Correlation coefficients for the sorted values of each column from the real dataset and its mirror column in the fake dataset. Higher is better. Bold indicates best in row, while italics indicate improvement over TGAN.

	TGAN	WGAN	SKIP	MedGAN	TableGAN
Berka	**0.9929**	0.9807	0.8640	0.4168	0.8899
Census	0.9854	**0.9871**	0.9840	-292.1 (0.9114)	0.9673
Creditcard	0.7999	**0.9817**	*0.9712*	-0.8533	0.1696

Table 7.5: Scores for the estimator results. Higher is better. Bold indicates best in row, while italics indicate improvement over TGAN. Berka and census are $1 - \text{MAPE}$, while creditcard is the correlation coefficient of RMSE scores.

approach taken is to predict the transaction amount, making it a regression problem instead of a classification problem.

We observe that TGAN-WGAN-GP improves upon TGAN on all three datasets, with TGAN-skip improving upon TGAN in the Berka and creditcard dataset. TableGAN does not improve on TGAN and neither does MedGAN. Additionally, something interesting happens with MedGAN in the census dataset. The logistic regression and MLP classifiers both perform so poorly on the real dataset when trained on the fake dataset, that the MAPE score skyrockets far beyond 1. Taking $1-\text{MAPE}$ results in a large negative number. We also observed large swings in this MAPE score specifically, indicating that the random sampling of test set seems to matter a lot. This is likely related to the fact that the census dataset has a lot of duplicate rows, and how these rows are distributed in the train and test set influences the F1-scores. Removing these two outlier values, we get the value in brackets, which is a lot more reasonable. However, in the final evaluation we use the original value.

7.6 Privacy Evaluations

We evaluate two privacy metrics: the number of rows in the synthetic dataset that occur in the real dataset and the mean and standard deviation to the closest record from the fake dataset to the real dataset. From a privacy perspective, it is best to have a large mean and a small standard deviation, but from a synthetic data perspective you want them to be as close to 0 as possible. Furthermore, the magnitude of the mean and standard deviation correlate with the number of columns of the tables due to the way they are calculated, so comparisons are only fair within a dataset.

	TGAN		WGAN		SKIP		MedGAN		TableGAN	
	Dist.	Copies	Dist.	Copies	Dist.	Copies	Dist.	Copies	Dist.	Copies
Berka	0.27±0.20	0	0.34±0.31	0	0.29±0.22	0	1.78±0.47	0	0.49±0.44	0
Census	2.39±1.45	0	2.69±1.24	5	2.83±1.28	2	5.51±1.19	0	2.86±1.32	1351
Creditcard	2.75±2.04	0	2.63±1.86	0	2.65±1.70	0	4.21±2.33	0	2.60±1.85	0

Table 7.6: The first value indicates the distances (mean + standard deviation) for all rows in the fake dataset and their most similar row in the real dataset. The second indicates the number of rows that occur exactly in the real dataset, i.e. are copies.

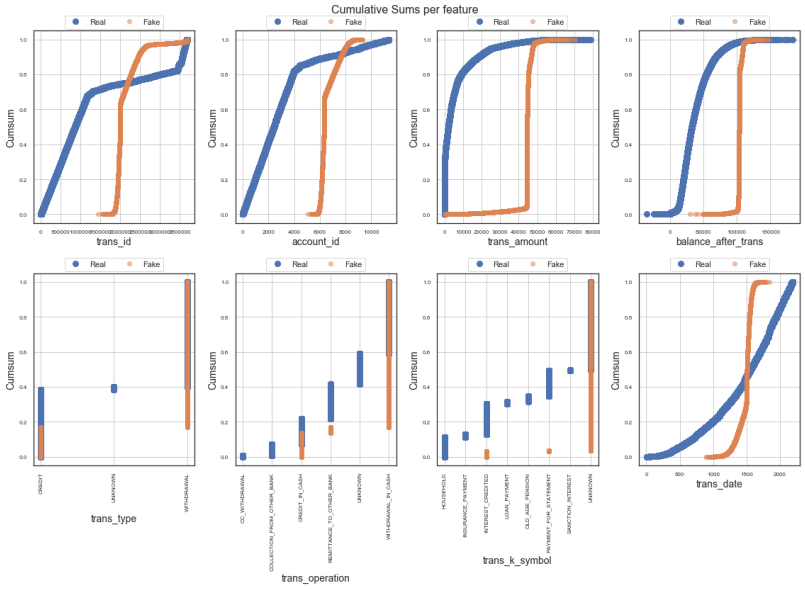
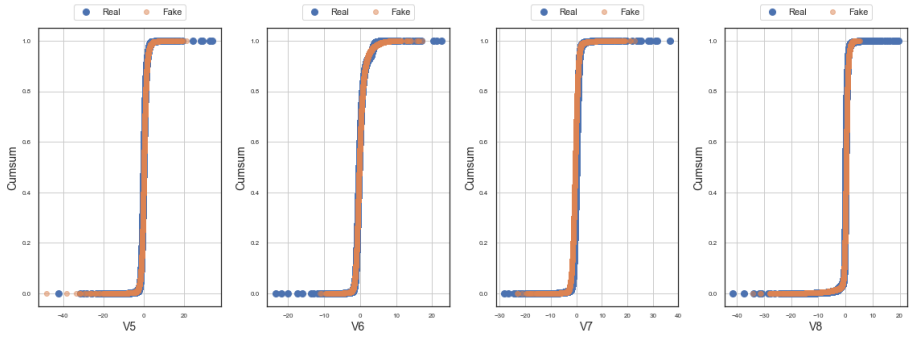
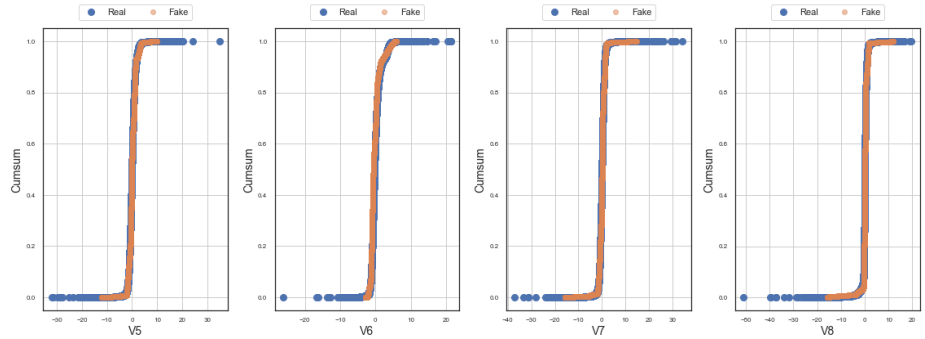


Figure 7.5: Cumulative sums of MedGAN on the Berka dataset. The distributions clearly indicate mode collapse is occurring.

We also see that most synthesizers do not replicate any data from the real dataset, with TableGAN having a surprising amount of duplicating in the Census Dataset. It seemed it suffered from partial mode collapse, because most of the 1351 copies are the same. We see that TGAN-WGAN-GP and TGAN-skip both contain a couple of copies, with can be attributed to getting closer to the real dataset that eventually you will generate data that appears in the real dataset. At this moment, the number of copied data points is not a part of the final similarity metric, but this might be a valuable addition to it. Knowing there are over a thousand copied rows is very relevant information. Especially since TableGAN has the overall best performance on the census dataset. A metric like $1 - \lambda \frac{n_{copies}}{N}$ with n_{copies} being the number of copied rows, N being the number of rows in the dataset and λ being a parameter to control the impact, could help reflect the necessary change. A λ might be required, because without it, the 1351 copied rows would still give a score of 0.98649, resulting in a relatively low penalty.



(a) Cumulative sums of TableGAN on the creditcard dataset. Blue indicates real data points, orange synthetic.



(b) Cumulative sums of TGAN-WGAN-GP on the creditcard dataset. Blue indicates real data points, orange synthetic.

Figure 7.6: (a) shows the cumulative sums of four columns of the creditcard dataset created by TableGAN, while (b) does the same for TGAN-WGAN-GP. It becomes clear that TableGAN is better in capturing values further from the mean, reaching up into the tails of the distribution. TGAN and its variants do this much less.

7.7 Visual Evaluations

The visual inspection of results provides some insight that the quantitative results alone do not. For example, it allows us to see what is happening with MedGAN and why its results are quite low. In figure 7.5 we observe the column distributions of MedGAN on the Berka dataset. It becomes clear that MedGAN is only synthesizing samples that have values within certain ranges. This indicates that mode collapse is occurring, and the generator is successfully fooling the discriminator with a subset of the real data. Additionally, we can see where the differences between the TGAN versions and TableGAN lie. This is presented in figure 7.6.

The hypothesis for the cause lies in the generator creating the intermediate representation for the autoencoder. As discussed in section 3.7, an autoencoder is not suitable for arbitrary latent space sampling, because the latent space contains gaps. In this situation, it would find some very good latent vectors which map to almost real data reconstructions. We tried to solve this by creating a continuous latent space using a VAE instead of a normal autoencoder. The VAE has a continuous latent space that allows for arbitrary sampling. However, the same pattern occurred and thus the results are fully attributed to mode collapse.

Figure 7.6 presents the differences between continuous column distributions from TableGAN and TGAN-WGAN-GP. The other TGAN variants follow the same pattern. This shows us why TableGAN get much higher scores for S_{pca}. This can be explained by the fact that the TGAN variants normalize with two times the standard deviation, which captures 97.7%, whereas TableGAN uses four times the standard deviation which captures 99.99%. Both are then clipped to $[-0.99, 0.99]$. This $\pm 2.3\%$ difference likely explains why TGAN does not get into the tails as much.

Since these cumulative sum plots are mainly useful to indicate some identified characteristic, are not part of the quantitative analysis and come in great numbers (± 450), we refrained from including all of them here or in the appendix. If interested to see these, please contact us.

7.8 Similarity Scores

Finally, we calculate the similarity scores, presented in table 7.7. The pattern of earlier results are repeated, with TGAN performing best in the Berka dataset while TGAN-WGAN-GP is best in creditcard and census. Furthermore, TGAN-skip and TableGAN also outperform TGAN in the census and creditcard dataset. MedGAN performs by far the worst, suffering from mode collapse. Note that the samples it does generate are realistic by themselves, just that the whole synthetic dataset characteristics are not an accurate approximation of the real data.

Going back to the original hypotheses of using the WGAN-GP architecture and skip-connection, it is fair to say that both improve performance over TGAN. In the cases they did not, their results are quite close to those of TGAN, like in the Berka dataset. When they improve on TGAN, it is often with fairly big margins.

To give some feel for the fake data, figure 7.8 contains a sample of real and fake data.

	TGAN	WGAN-GP	TGAN-SKIP	MedGAN	TableGAN
Berka					
basic statistics	0.9910	0.9955	0.9850	0.9113	0.9895
Correlation column correlations	0.9821	0.9470	0.9832	0.7694	0.6468
Mirror Column Correlation	0.9276	0.9150	0.9572	0.5602	0.8864
1 - MAPE Estimator	0.9929	0.9807	0.8640	0.4168	0.8899
1 - MAPE PCA	0.9456	0.9399	0.9424	0.8236	0.9465
Similarity Score	**0.9678**	0.9556	0.9464	0.6963	0.8718
Census					
basic statistics	0.9212	0.9909	0.9894	0.4325	0.9947
Correlation column correlations	0.9581	0.9773	0.9053	0.0644	0.9128
Mirror Column Correlation	0.7008	0.8722	0.7941	0.2092	0.8651
1 - MAPE Estimator	0.9854	0.9871	0.9840	-292.1544	0.9673
1 - MAPE PCA	0.9507	0.9739	0.9643	0.6907	0.9748
Similarity Score	0.9032	**0.9603**	*0.9274*	-58.1515	*0.9429*
Creditcard					
basic statistics	0.8028	0.8661	0.8799	-0.0329	0.8734
Correlation column correlations	0.0968	0.2932	0.2114	-0.0471	0.2157
Mirror Column Correlation	0.9215	0.9605	0.9342	0.7888	0.9425
Correlation RMSE	0.7999	0.9817	0.9712	-0.8533	0.1696
1 - MAPE PCA	0.8501	0.7810	0.7266	0.1726	0.8667
Similarity Score	0.6942	**0.7765**	*0.7447*	0.0056	0.6136

Table 7.7: All final metrics and the corresponding similarity score, which is the mean of the five other scores.

trans_id	account_id	trans_amount	balance_after_trans	trans_type	trans_operation	trans_k_symbol	trans_date
674640	2305	6000	243959	WITHDRAWAL	WITHDRAWAL_IN_CASH	UNKNOWN	742
2789525	9236	146	289345	WITHDRAWAL	WITHDRAWAL_IN_CASH	PAYMENT_FOR_STATEMENT	576
3674501	6138	748	192086	CREDIT	UNKNOWN	INTEREST_CREDITED	2037
435602	1480	34252	311763	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	LOAN_PAYMENT	1957
747762	2550	146	523926	WITHDRAWAL	WITHDRAWAL_IN_CASH	PAYMENT_FOR_STATEMENT	1611
337627	1146	13840	734047	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	UNKNOWN	1014
873155	2974	55820	378277	CREDIT	COLLECTION_FROM_OTHER_BANK	OLD_AGE_PENSION	892
1372411	4681	50160	394154	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	HOUSEHOLD	1528
1032713	3530	66000	111710	WITHDRAWAL	WITHDRAWAL_IN_CASH	UNKNOWN	2057
882616	3007	55960	452086	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	INSURANCE_PAYMENT	1164
680517	2322	146	315633	WITHDRAWAL	WITHDRAWAL_IN_CASH	PAYMENT_FOR_STATEMENT	1884

(a) Sample of the real data from the Berka data set

trans_id	account_id	trans_amount	balance_after_trans	trans_type	trans_operation	trans_k_symbol	trans_date
2774156	8978	62170	228850	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	LOAN_PAYMENT	2009
269721	645	22320	262990	WITHDRAWAL	WITHDRAWAL_IN_CASH	UNKNOWN	1810
3574794	224	340	200910	CREDIT	UNKNOWN	INTEREST_CREDITED	1270
1280872	4223	25940	186010	WITHDRAWAL	WITHDRAWAL_IN_CASH	UNKNOWN	1640
3227842	10455	26000	617300	WITHDRAWAL	WITHDRAWAL_IN_CASH	UNKNOWN	2059
2310514	7195	143350	716130	CREDIT	CREDIT_IN_CASH	UNKNOWN	1520
3612642	2291	960	350940	CREDIT	UNKNOWN	INTEREST_CREDITED	2057
1027721	3394	212400	1149170	CREDIT	CREDIT_IN_CASH	UNKNOWN	1942
407896	1157	1860	161090	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	UNKNOWN	883
3659869	3557	1000	469260	CREDIT	UNKNOWN	INTEREST_CREDITED	1162
1583806	5325	41220	227030	WITHDRAWAL	REMITTANCE_TO_OTHER_BANK	HOUSEHOLD	1627

(b) Sample of the fake data from the Berka data set, synthesized by TGAN.

Table 7.8: Sample (a) is a real sample while sample (b) is a synthetic sample. It gives you some feel as to how similar they can be. Both samples where randomly sampled from the total dataset.

Chapter 8

Conclusions

In this thesis, we have shown that considerable improvements can be made in the generation of synthetic data when using skip-connections and the WGAN-GP architecture. Both adaptations to the TGAN model displayed an improvement in sample quality over TGAN, while also improving on other models like MedGAN and TableGAN. Additionally, we have determined that the usage of synthetic tabular data, created by GANs are a serious contender for usage in industry applications like finance, healthcare and government applications.

Furthermore, We have proposed an evaluation method that provides a single value indicator for the similarity of a synthetic dataset when compared to a real dataset. This Similarity Score correlates with sample quality and allows for easy evaluation of synthetic data generation models. When used, this metric results in more consistent and insightful evaluation of synthetic data than previous methods.

Future improvements can be achieved in a couple of different areas. First, we did not explore combining the TGAN-WGAN-GP architecture *and* skip-connections at the same time, which would be a good next step. Also, optimizing the hyper parameters of the normalization might yield improvements, as we concluded that some implementations limit the distribution of generated data. The way the PCA metric was evaluated was unfair compared to models that had stronger performance in categorical data, rather than in continuous data. Using a more balanced metric could improve the stability of results. Finally, we should consider experiments with larger datasets and longer training periods, since the performance of TGAN on the Berka dataset indicated that convergence might take longer than the 100 epochs used by us. As a last direction of research, methods that expand the number of data types can be explored. Dates and ordinal data are both suitable directions of research.

Bibliography

[1] ARX - data anonymization tool. http://arx.deidentifier.org.

[2] Credit card fraud dataset, 2013. URL `www.kaggle.com/mlg-ulb/creditcardfraud`.

[3] Auto-encoder: What is it? and what is it used for? (part 1), 2019. URL `https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726`.

[4] How to code gaussian mixture models from scratch in python, 2019. URL `https://towardsdatascience.com/how-to-code-gaussian-mixture-models-from-scratch-in-python-9e7975df5252`.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. 2017. URL `https://arxiv.org/pdf/1701.07875.pdf`.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[7] P. Berka and M. Sochorova. 1999 czech financial dataset - real anonymized transactions, 2019. URL `https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions`.

[8] D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[9] S. Bhattarai. What is gradient descent in machine learning, 2018. URL `https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/`.

[10] O. Carey. Generative adversarial networks (gans) — a beginner's guide, 2018. URL `https://towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-5b38eceece24`.

[11] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[12] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart, S. Org, and J. Sun. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. URL `https://arxiv.org/pdf/1703.06490.pdf`.

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

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. URL `https://arxiv.org/abs/1810.04805`.

[16] Y. Dodge. *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008.

[17] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[18] R. A. Fisher. *Statistical methods for research workers*. 1926.

[19] C. Glosser. Artificial neural network with layer coloring, 2013. URL `https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg`. [Online; accessed October 1, 2019].

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. URL `http://www.github.com/goodfeli/adversarial`.

[21] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[22] M. Greenacre and J. Blasius. *Multiple correspondence analysis and related methods*. Chapman and Hall/CRC, 2006.

[23] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs. 2017. URL `http://arxiv.org/abs/1704.00028`.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. URL `http://image-net.org/challenges/LSVRC/2015/`.

[25] D. O. Hebb. *The organization of behavior: a neuropsychological theory*. Science Editions, 1962.

[26] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. Technical report. URL `https://arxiv.org/pdf/1706.08500.pdf`.

[27] G. Hinton and V. Nair. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. URL `https://arxiv.org/abs/1608.06993`.

[29] J. Hui. Gan — wasserstein gan & wgan-gp, 2019. URL `https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490`.

[30] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[31] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[32] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. 2016. URL `https://arxiv.org/pdf/1611.01144.pdf`.

[33] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. *arXiv e-prints*, art. arXiv:1611.01144, Nov 2016.

[34] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[36] S. K. Kinney, J. P. Reiter, A. P. Reznek, J. Miranda, R. S. Jarmin, and J. M. Abowd. Towards unrestricted public use business microdata: The synthetic longitudinal business database. *International Statistical Review / Revue Internationale de Statistique*, 79 (3):362–384, 2011. ISSN 03067734, 17515823. URL `http://www.jstor.org/stable/41305056`.

[37] N. Kodali, J. Abernethy, J. Hays, and Z. Kira. On convergence and stability of gans. *arXiv preprint arXiv:1705.07215*, 2017.

[38] F. Kratzert. Understanding the backward pass through batch normalization layer, 2016. URL `https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html`.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[40] T. Lane and R. Kohavi. Census-income (kdd) data set, 2000. URL `https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29`.

[41] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[42] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[43] Z. Lin, A. Khetan, G. Fanti, and S. Oh. PacGAN: The power of two samples in generative adversarial networks. URL `https://arxiv.org/pdf/1712.04086.pdf`.

[44] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal. *A Large-Scale Study. ArXiv e-prints*, 2(4), 2017.

[45] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[46] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[47] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks, 2016.

[48] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[49] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[50] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim. Data Synthesis based on Generative Adversarial Networks. *Data Synthesis based on Generative Adversarial Networks. PVLDB*, 11(10):1071–1083, 2018. doi: 10.14778/3231751.3231757. URL `http://arxiv.org/abs/1806.03384`.

[51] N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410. IEEE, 2016.

[52] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[53] I. Shafkat. Intuitively understanding variational autoencoders - towards data science, Apr 2018. URL `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`.

[54] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[55] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

[56] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[57] M. Templ, A. Kowarik, and B. Meindl. Statistical disclosure control for micro-data using the R package sdcMicro. *Journal of Statistical Software*, 67(4):1–36, 2015. doi: 10.18637/jss.v067.i04.

[58] L. Xu and K. Veeramachaneni. Synthesizing Tabular Data using Generative Adversarial Networks. 2018. URL `http://arxiv.org/abs/1811.11264`.

[59] L. Xu, M. Skoularidou, and A. Cuesta-infante. Modeling Tabular data using Conditional GAN. pages 1–24. URL `https://github.com/DAI-Lab/SDGym`.

[60] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):25, 2017.

[61] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Appendix A

Appendix

A.1 Results

A.1.1 Basic Evaluation

In table A.1 the correlations between the basic statistical values are presented. TGAN-WGAN-GP is again the strongest in, outperforming TGAN in all tasks. However, it is only the best in Berka, with TableGAN being best in the creditcard dataset and Skip being best in the census dataset. MedGAN does not perform well. I did not include all the data the basic evaluation results were based on due to space constraint. The number of values would be $\sum_{ds \in DS} \sum_{m \in M} 2 \cdot ds_{columns}$ where DS is the set of all the datasets, M is the set of all synthesization models used and $ds_{columns}$ is the number of columns in ds. This amounts to 2340 values. If you would like to see these values, please contact me.

	TGAN	WGAN-GP	TGAN-skip	MedGAN	TableGAN
Berka	0.9909	**0.9954**	0.9849	0.9112	0.9894
Census	0.9212	*0.9909*	*0.9893*	0.4324	**0.9947**
Creditcard	0.8028	*0.8661*	**0.8799**	-0.0329	*0.8734*

Table A.1: Basic statistics correlation coefficients for the real and synthetic dataset. Higher is better. Bold indicates best in row, while italics indicate improvement over TGAN.

A.1.2 Column correlations

In figure A.2, the column correlation for all the creditcard datasets are plotted.

A.1.3 Machine learning efficacy

The machine learning efficacy results omitted in the results section are presented in figure A.4. These are included to give some insights into the specifics of the machine learning efficacy of all the methods, which are one of the most interesting results.

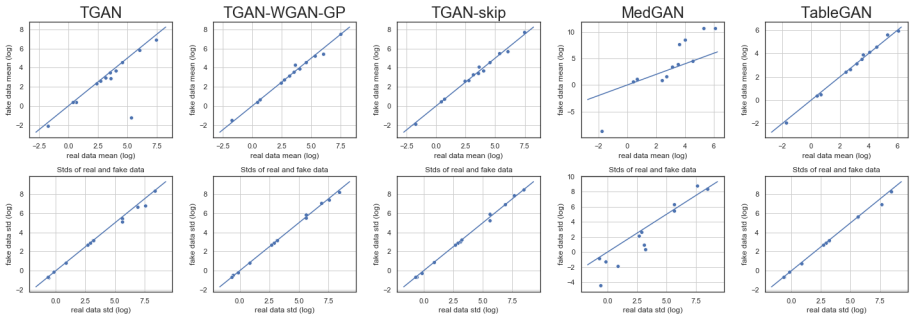


Figure A.1: Mean and standard deviations of each column of the real and synthetic census dataset. All values are log transformed.

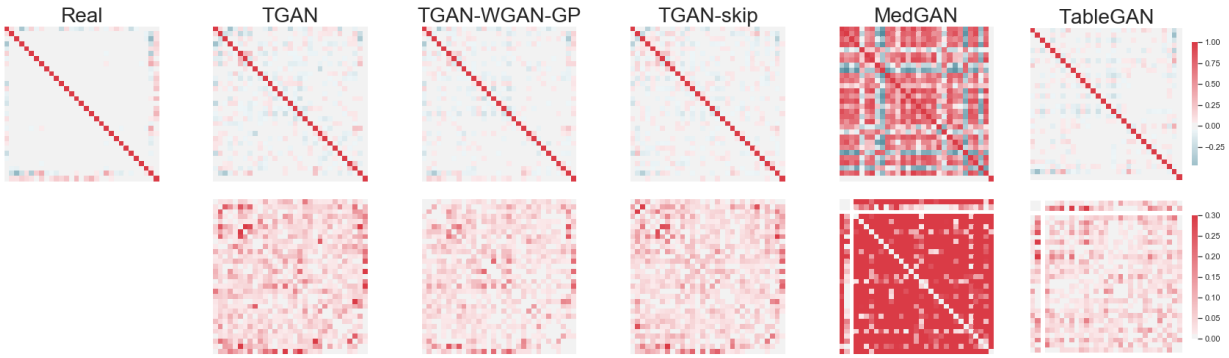


Figure A.2: Associations per column from the real dataset and each of the synthesizers on the Creditcard dataset. The bottom row is the absolute difference between the synthesizer and the ground truth. The range of the difference is reduced to $[0, 0.3]$.

	Logistic Regression		Random Forest		Decision Tree		MLP	
Trained on	Real	Fake	Real	Fake	Real	Fake	Real	Fake
Evaluated on								
TGAN								
Real	0.7988	0.7938	0.9852	0.9808	0.9761	0.9685	0.7904	0.7697
Fake	0.7956	0.7916	0.9833	0.9810	0.9576	0.9667	0.6010	0.5994
TGAN-WGAN-GP								
Real	0.7932	0.7247	0.9832	0.9248	0.9745	0.9145	0.6011	0.6003
Fake	0.7778	0.7292	0.9719	0.9398	0.9126	0.9085	0.6005	0.6003
TGAN-skip								
Real	0.7937	0.7186	0.9841	0.9696	0.9740	0.9593	0.5976	0.5668
Fake	0.7861	0.7469	0.9829	0.9702	0.9326	0.9460	0.5905	0.5666
MedGAN								
Real	0.8062	0.1688	0.9849	0.9637	0.9770	0.9637	0.8318	0.1688
Fake	0.7614	0.8655	0.7789	0.9981	0.9614	0.9970	0.5735	0.8312
TableGAN								
Real	0.7984	0.7170	0.9835	0.8192	0.9744	0.8086	0.7901	0.7134
Fake	0.7805	0.7024	0.9043	0.9103	0.8700	0.8802	0.7193	0.6754

Table A.2: F1 scores for each classifier on the Berka dataset.

	Logistic Regression		Random Forest		Decision Tree		MLP	
Trained on	Real	Fake	Real	Fake	Real	Fake	Real	Fake
Evaluated on								
TGAN								
Real	0.9511	0.9556	0.9503	0.9567	0.9339	0.9314	0.9443	0.9367
Fake	0.9425	0.9595	0.9406	0.9580	0.9032	0.9321	0.9385	0.9567
TGAN-WGAN-GP								
Real	0.9473	0.9301	0.9489	0.9292	0.9297	0.9056	0.9489	0.9289
Fake	0.9412	0.9302	0.9388	0.9305	0.8999	0.9028	0.9341	0.9298
TGAN-skip								
Real	0.9513	0.9265	0.9508	0.9283	0.9341	0.8902	0.9533	0.9155
Fake	0.9401	0.9294	0.9388	0.9294	0.8870	0.8809	0.9384	0.9290
MedGAN								
Real	0.9468	0.0013	0.9468	0.9711	0.9308	0.8093	0.9372	0.0013
Fake	0.9281	0.9998	0.9349	0.9994	0.6077	0.9992	0.9341	0.9987
TableGAN								
Real	0.9502	0.9305	0.9478	0.9299	0.9325	0.9214	0.9424	0.9341
Fake	0.9295	0.9644	0.9372	0.9654	0.9260	0.9624	0.9293	0.9688

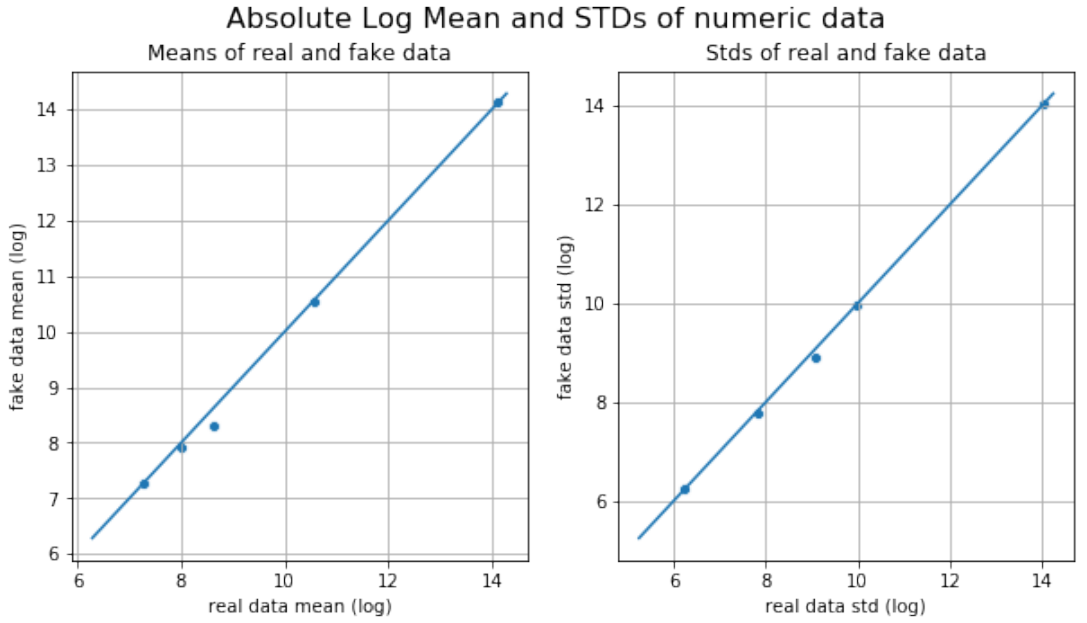
Table A.3: F1 scores for each classifier on the census dataset.

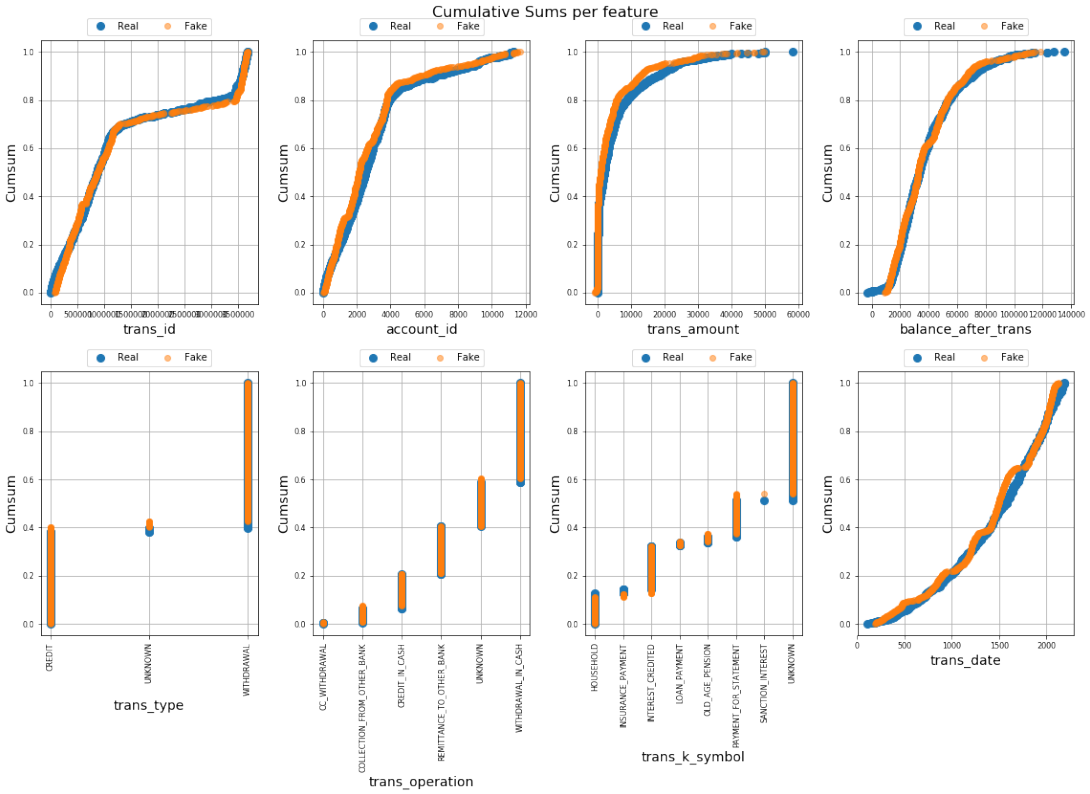
	Random Forest		Lasso		Ridge		ElasticNet	
Trained on	Real	Fake	Real	Fake	Real	Fake	Real	Fake
Evaluated on								
TGAN								
Real	82	320	74	327	75	327	95	336
Fake	232	247	207	298	212	298	148	315
TGAN-WGAN-GP								
Real	83	223	68	246	67	248	94	227
Fake	150	207	124	219	123	219	147	221
TGAN-skip								
Real	99	232	74	254	74	255	100	236
Fake	189	215	138	229	136	229	163	232
MedGAN								
Real	83	7,896	74	5,300	74	5,033	93	7,175
Fake	13,148	47	10,669	32	8,263	12	9,976	69
TableGAN								
Real	83	120	63	123	63	124	87	121
Fake	106	247	78	110	91	111	100	121

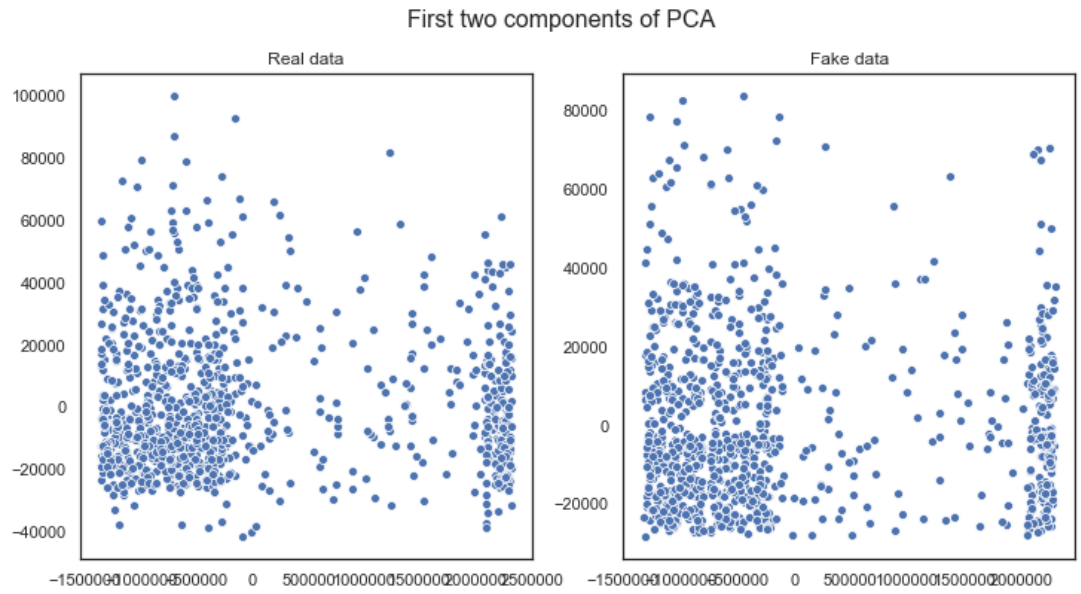
Table A.4: RMSE scores for each regressor on creditcard dataset. Each synthesizer seems to capture relations well enough for the RMSE to be fairly close, except for MedGAN. In the case of MedGAN, we see a strong divergence between the real and fake dataset MRSE.

A.2 Python Package Example: TableEvaluator

```
In [1]: from table_evaluator import *

In [2]: real, fake = load_data('data/real_test_sample.csv', 'data/fake_test_sample.csv')

In [3]: real.head()

Out[3]:    trans_id  account_id  trans_amount  balance_after_trans  trans_type  \
        0    951892        3245        3878.0              13680.0  WITHDRAWAL
        1   3547680         515          65.9              14898.6      CREDIT
        2   1187131        4066       32245.0              57995.5      CREDIT
        3    531421        1811        3990.8              23324.9  WITHDRAWAL
        4     37081         119       12100.0              36580.0  WITHDRAWAL

                     trans_operation     trans_k_symbol  trans_date
        0    REMITTANCE_TO_OTHER_BANK          HOUSEHOLD        2165
        1                     UNKNOWN  INTEREST_CREDITED        2006
        2  COLLECTION_FROM_OTHER_BANK            UNKNOWN        2139
        3    REMITTANCE_TO_OTHER_BANK       LOAN_PAYMENT         892
        4          WITHDRAWAL_IN_CASH            UNKNOWN         654

In [4]: fake.head()

Out[4]:    trans_id  account_id  trans_amount  balance_after_trans  trans_type  \
        0    911598        3001       13619.0              92079.0      CREDIT
        1    377371        1042        4174.0              32470.0  WITHDRAWAL
        2    970113        3225         274.0              57608.0  WITHDRAWAL
        3    450090        1489         301.0              36258.0      CREDIT
        4   1120409        3634        6303.0              50975.0  WITHDRAWAL

                     trans_operation trans_k_symbol  trans_date
        0  COLLECTION_FROM_OTHER_BANK        UNKNOWN        1885
        1    REMITTANCE_TO_OTHER_BANK      HOUSEHOLD        1483
        2          WITHDRAWAL_IN_CASH        UNKNOWN        1855
        3              CREDIT_IN_CASH        UNKNOWN         885
        4    REMITTANCE_TO_OTHER_BANK      HOUSEHOLD        1211

In [5]: cat_cols = ['trans_type', 'trans_operation', 'trans_k_symbol']

In [6]: table_evaluator = TableEvaluator(real, fake, cat_cols=cat_cols)

In [7]: table_evaluator.visual_evaluation()
```


Absolute Log Mean and STDs of numeric data
Means of real and fake data
fake data mean (log)
real data mean (log)
Stds of real and fake data
fake data std (log)
real data std (log)


Cumulative Sums per feature
Real
Fake
Cumsum
trans_id
account_id
trans_amount
balance_after_trans
trans_type
CREDIT
UNKNOWN
WITHDRAWAL
trans_operation
CC_WITHDRAWAL
COLLECTION_FROM_OTHER_BANK
CREDIT_IN_CASH
REMITTANCE_TO_OTHER_BANK
UNKNOWN
WITHDRAWAL_IN_CASH
trans_k_symbol
HOUSEHOLD
INSURANCE_PAYMENT
INTEREST_CREDITED
LOAN_PAYMENT
OLD_AGE_PENSION
PAYMENT_FOR_STATEMENT
SANCTION_INTEREST
UNKNOWN
trans_date




```
In [8]: table_evaluator.evaluate(target_col='trans_type')


Correlation metric: pearsonr

Classifier F1-scores:
                                    real   fake
real_data_LogisticRegression_F1     0.8550 0.8450
real_data_RandomForestClassifier_F1 0.9950 0.9950
real_data_DecisionTreeClassifier_F1 0.9700 0.9450
real_data_MLPClassifier_F1          0.3950 0.6250
fake_data_LogisticRegression_F1     0.7750 0.7800
fake_data_RandomForestClassifier_F1 0.9600 0.9550
fake_data_DecisionTreeClassifier_F1 0.9450 0.9500
```

```
fake_data_MLPClassifier_F1          0.4500 0.5550

Miscellaneous results:
                                        Result
Column Correlation Distance RMSE        0.0399
Column Correlation distance MAE         0.0296
Duplicate rows between sets (real/fake) (0, 0)
nearest neighbor mean                   0.5655
nearest neighbor std                    0.3726

Results:
                                                Result
basic statistics                                0.9940
Correlation column correlations                 0.9904
Mean Correlation between fake and real columns  0.9566
1 - MAPE Estimator results                      0.8912
1 - MAPE 5 PCA components                       0.9138
Similarity Score                                0.9492
```